

TECHNISCHE UNIVERSITÄT MÜNCHEN

Fakultät für Informatik  
I-16 / Computer Aided Medical Procedures

# Accelerated Registration and Reconstruction for Functional Nuclear Imaging

Christoph Philipp Friedrich Joachim Vetter

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. Helmut Seidl

Prüfer der Dissertation:

1. Univ.-Prof. Dr. Nassir Navab
2. Univ.-Prof. Dr. Rüdiger Westermann

Die Dissertation wurde am 20. Oktober 2015 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 10. Februar 2016 angenommen.



---

## Abstract

Functional nuclear imaging provides unique information that is complementary to structural imaging modalities like CT or MR which provide a view of the patient's anatomy. Functional imaging modalities like SPECT or PET on the other hand provide a view into physiological activities, whether that is metabolic function, blood flow or the chemical composition of tissues. This information is widely used for diagnosis, but also very valuable in the interventional case.

Time constraints may make the application of computationally expensive algorithms unfeasible in clinical settings. This is true in diagnostic settings, but the time constraints are even stricter for intra-operative purposes. The tightly orchestrated workflow in the operating room (often involving a large support staff) does not allow for costly down times, but the information provided during surgery becomes quickly inaccurate or even invalid because of tissue manipulation at the same time. So while time is constrained, there is a constant need for updated information.

This thesis develops accelerated methods for the registration and reconstruction of functional nuclear volumes, making the application of these algorithms feasible in clinical practice. The first chapters develop the fundamental principles of registration and algorithms for massively parallel hardware that are expanded upon in later chapters for the development of the methods introduced in this thesis. In order to achieve the required interactive speed, hardware-based acceleration of existing algorithms is one option, the introduction of novel algorithms that replace slower ones the second option. This dissertation demonstrates both approaches.

The contributions of this dissertation include one of the first registration pipelines for deformable registration fully accelerated by graphics processing units (GPUs). Using statistical intensity priors, this method allows the accurate and fast registration between different modalities. The next contribution is the careful optimization of the main bottleneck for registration methods using statistical intensity priors on massively parallel hardware architectures like GPUs: The optimization of the joint histogram computation with its parallelization unfriendly memory access patterns. The third contribution is a GPU-accelerated pipeline for SPECT reconstruction.

The main contribution of this thesis lies in a new registration method between a 3D volume (acquired pre- or intraoperatively) and intra-operative tracked probe measurements for radio-guided surgery. The registration allows rapid updates of the image guidance for the surgeon, while only requiring little time. The usefulness of this approach is demonstrated for the case of intra-operative freehand SPECT reconstruction during sentinel lymph node biopsies for breast cancer treatment. Using a previous 3D volume as prior knowledge, the 1D-3D registration allows the fast update of reconstruction volumes by performing a registration to 1D measurements provided by a gamma probe. This replaces in effect a reconstruction step with registration using the prior knowledge. Finally, we provide an outlook towards further research necessary in order to make this a viable alternative to current techniques.

---

## Zusammenfassung

Funktionale nukleare Bildgebungsverfahren stellen einzigartige Informationen zur Verfügung und komplementieren Bildgebungsverfahren für Strukturen wie CT oder MR, die eine Sicht auf die Anatomie des Patienten gewähren. Funktionale Bildgebungsverfahren wie SPECT oder PET visualisieren stattdessen physiologische Aktivitäten, seien das Funktionen des Stoffwechsels, des Blutflusses oder der chemischen Zusammensetzung von Gewebe. Diese Information wird weithin für die Diagnose benutzt, ist aber auch wertvoll für operative Eingriffe.

Zeitbeschränkungen machen die Anwendung von rechenaufwändigen Algorithmen unpraktikabel in klinischen Umgebungen. Dies trifft ebenfalls auf die Diagnose zu, aber die Zeitbeschränkungen für intra-operative Anwendungen sind deutlich strikter. Der genau choreographierte Arbeitsablauf in dem Operationssaal (der oft eine große Anzahl von unterstützendem Personal erfordert), erlaubt keine teuren Unterbrechungen, aber aufgrund der Manipulation des Gewebes des Patienten werden die Information, die zur Verfügung stehen, während des Eingriffs schnell ungenau oder ungültig. Auch wenn die Zeit, die zur Verfügung steht, beschränkt ist, besteht ein dauernder Bedarf an aktualisierter Information.

Diese Arbeit entwickelt beschleunigte Methoden für die Registrierung und Rekonstruktion von funktionalen nuklearen Volumen, um die Anwendung dieser Algorithmen in der Praxis zu erleichtern. Die ersten Kapitel entwickeln die grundlegenden Prinzipien der Registrierung sowie von Algorithmen auf massiv paralleler Hardware. Auf diesen Grundlagen aufbauend, werden die neu eingeführten Methoden in dieser Dissertation vorgestellt.

Die Hardware-Beschleunigung von existierenden Algorithmen ist eine Möglichkeit, um die benötigte interaktive Geschwindigkeit zu erreichen, die Einführung neuer schnellerer Algorithmen ist eine zweite Möglichkeit. Die Dissertation demonstriert beide Möglichkeiten.

Der Beitrag dieser Arbeit beinhaltet eine der ersten voll GPU-beschleunigten Registrierungen deformierbarer Volumen. Vorwissen über statistische Intensitätsverteilungen ermöglicht die genaue und schnelle Registrierung zwischen verschiedenen Modalitäten. Der nächste Beitrag ist die sorgfältige Optimierung des Hauptflaschenhalses für diese Registrierungsmethode auf massiv paralleler Hardware wie GPUs: die Optimierung der Berechnung der statistischen Intensitätsverteilung zweier Volumen mit ihrem Speicherzugriffsmustern, die ungünstig für die Parallelisierung sind. Der dritte Beitrag liegt in der GPU-beschleunigten Rekonstruktion von SPECT-Volumen.

Der Hauptbeitrag dieser Dissertation liegt in einer neuen Registrierungsmethode zwischen einem 3D-Volumen (vor oder während der Operation aufgenommen) und intra-operativen Messungen von einem Nukleardetektor und assoziierten Positionsdaten. Die Registrierung ermöglicht die schnelle Aktualisierung der Bildgebung für den Chirurgen, aber benötigt nur wenig Zeit. Die Nützlichkeit dieses Ansatzes wird für den Fall der intra-operativen freehand SPECT Rekonstruktion während einer Wächterknotenbiopsie für die Brustkrebsbehandlung demonstriert. Mit einem vorherigen 3D Volumen als Vorwissen erlaubt die 1D-3D Registrierung das schnelle Aktualisieren des Rekonstruktionsvolumen indem eine Registrierung mit den 1D Messungen des Gammadetektors durchgeführt wird. Dies ersetzt den Rekonstruktionsschritt durch einer Registrierung mit Vorwissen. Zum Abschluss geben wir noch

---

eine Aussicht auf weitere Forschungsbereiche, die notwendig sind, um diesen Ansatz zu einer praktikablen Alternative für die gegenwärtigen Vorgehensweisen zu machen.



# Contents

<b>Acknowledgements</b>	<b>ix</b>
<b>I. Introduction</b>	<b>1</b>
1. Thesis Overview	3
<b>II. State of the Art</b>	<b>7</b>
<b>2. Medical Image Registration</b>	<b>9</b>
2.1. Components of Image Registration	9
2.2. Image Representation	11
2.3. Interpolation	11
2.4. Transformation	13
2.4.1. Affine Transformations	14
2.4.2. Deformable Transformations	15
2.5. Similarity measures	16
2.5.1. Sum of Squared Distances	17
2.5.2. Sum of Absolute Distances	17
2.5.3. Normalized Cross Correlation	18
2.5.4. Mutual Information	18
2.5.5. Kullback-Leibler Divergence	19
2.5.6. Other Similarity Measures	19
2.6. Optimization Procedure	19
2.6.1. Best-Neighbor Search	22
2.6.2. Simplex Method	22
2.6.3. Powell-Brent Method	23
2.6.4. Gradient Descent Optimization	23
2.7. Regularization	23
2.8. Conclusion	25
<b>3. GPUs for High-Performance Computing</b>	<b>27</b>
3.1. Computing Coprocessors	27
3.2. Rendering as a Parallel Problem	28
3.3. Evolution of the GPU	28
3.4. The Graphics Pipeline	30
3.5. Comparison of the GPU and CPU architecture	32
3.6. High Performance Computing on the GPU	34
3.6.1. Using Graphics Libraries for General Purpose Computing	34
3.6.2. Programming Languages for the GPU	34
3.6.3. The CUDA Programming Model	37
3.7. Acceleration Evaluation	39
3.8. Outlook	39
3.9. Conclusion	41

<b>4. GPU-Accelerated Registration</b>	<b>43</b>
4.1. Interpolation	43
4.2. Transformation	43
4.2.1. Projection	43
4.2.2. Non-Rigid Transformation	43
4.3. Similarity Measures on the GPU	45
<b>III. Contributions</b>	<b>47</b>
<b>5. Learning-Based Registration</b>	<b>49</b>
5.1. Introduction: SPECT/CT Misregistration in Hybrid Scanners	49
5.2. Method	49
5.2.1. Statistical Intensity Priors	50
5.2.2. Cost Function	50
5.3. GPU Registration Pipeline	51
5.3.1. Histogram Computation on the GPU	52
5.3.2. Regularization Using Gaussian Smoothing on the GPU	54
5.3.3. Hybrid Implementation	55
5.3.4. GPU Optimization	55
5.3.5. Optimized Histogram Computation	56
5.4. Results	57
5.4.1. GPU-CPU Comparison	57
5.4.2. Performance Optimization	58
5.4.3. Integrated Rendering	59
5.5. Conclusion	60
<b>6. Joint Histograms</b>	<b>61</b>
6.1. Introduction	61
6.1.1. Computing the Joint Histogram	61
6.1.2. Computing Histograms on the GPU	61
6.1.3. Compute APIs for Histograms	62
6.2. Method	63
6.2.1. Preprocessing	64
6.2.2. The Histogram Kernel	64
6.2.3. Reducing Conflicts between Atomic Functions	67
6.3. Results	68
6.3.1. Memory Consumption	68
6.3.2. Random Read Patterns and Shared Memory Collisions	69
6.3.3. Timings	69
6.3.4. Registration	73
6.4. Conclusion	73
<b>7. SPECT Reconstruction on the GPU</b>	<b>75</b>
7.1. Introduction	75
7.2. Method	76
7.2.1. Radon Transform	76
7.2.2. Series Expansion Methods	76
7.2.3. Maximum Likelihood Expectation Maximization	77
7.2.4. Ordered Subset Expectation Maximization	78
7.2.5. Attenuation Correction	80
7.2.6. The Point Spread Function	80



7.3. GPU Implementation . . . . .	81
7.3.1. Previous Work . . . . .	81
7.3.2. Implementation . . . . .	82
7.3.3. Modeling the Point Spread Function . . . . .	82
7.3.4. Attenuation Modelling . . . . .	82
7.3.5. Forward Projection . . . . .	83
7.3.6. Projection Correction . . . . .	84
7.3.7. Backward Projection . . . . .	84
7.3.8. Volume Update . . . . .	84
7.3.9. Convolution . . . . .	85
7.4. Results . . . . .	85
7.5. Conclusion . . . . .	85
<b>8. 1D–3D Registration</b>	<b>87</b>
8.1. Introduction . . . . .	87
8.1.1. Clinical Problem . . . . .	87
8.1.2. Intra–operative Guidance and Interventional Imaging . . . . .	89
8.2. Methods . . . . .	90
8.2.1. Surgical Workflow . . . . .	90
8.2.2. Input Data and Output . . . . .	91
8.2.3. Registration Procedure . . . . .	91
8.2.4. Detector Model . . . . .	92
8.2.5. Node Segmentation . . . . .	92
8.2.6. Decay Correction . . . . .	93
8.2.7. The Registration Pipeline for Intra–operative and Pre–operative SPECT . . . . .	93
8.3. Experiments . . . . .	94
8.3.1. Monte–Carlo Experiments . . . . .	94
8.3.2. Phantom Experiments . . . . .	95
8.3.3. Intra–operative Data . . . . .	96
8.4. Results . . . . .	97
8.4.1. Monte–Carlo Experiments . . . . .	97
8.4.2. Phantom Experiments . . . . .	99
8.4.3. Intra–operative Data . . . . .	100
8.5. Discussion . . . . .	101
8.5.1. Monte–Carlo Experiments . . . . .	101
8.5.2. Phantom Experiments . . . . .	101
8.5.3. Intra–operative Data . . . . .	102
8.5.4. Future Directions . . . . .	103
8.6. Conclusion . . . . .	104
<b>IV. Appendix</b>	<b>107</b>
<b>List of Figures</b>	<b>109</b>
<b>List of Tables</b>	<b>111</b>
<b>A. Acronyms</b>	<b>113</b>
<b>B. List of Publications</b>	<b>115</b>
<b>Bibliography</b>	<b>117</b>



# Acknowledgements

I would like to thank my adviser Nassir Navab for the opportunity to finish my Ph.D. at the CAMP chair, as well as for his guidance and support during the last five years of my dissertation. Ali Kamen and Oliver Kutter thanks for providing the contact to Nassir!

I am very thankful to Tobias Lasser for his support and help with my nuclear imaging papers, as well as his comments on my thesis. José Gardiazabal and Tobias Reichl provided much support for my robotic experiments, even if this research never ended up in a published paper.

My first adviser Rüdiger Westermann as well as my various co-authors especially Christoph Guetter, Chenyang Xu, Zhe Fan, Daphne Yu, Ali Kamen, Parmeshwar Khurd, Oliver Fluck, Wolfgang Wein, Thomas Wendler, Ash Okur helped me a lot on my long path to my Ph.D. I would also like to thank my colleagues at Siemens and my supervisors during that time Gianluca Paladinin and Daphne Yu. Luc and Michael have provided valuable feedback on my thesis. I would like to additionally thank my parents, as well as Nicolas, Michael and Marian for their moral support especially during the more frustrating times. Then, life would have been so much more boring without Amanda, Ren-Yi, Christoph, Kay, Sasa, Brian, Karin, Konrad and Tim.

Finally, I would like to especially thank Tobias Lasser and Manuela Fischer for their support during the 14-month long, kafkaesk “Papierkrieg” to finally allow me to officially submit my thesis.



**Part I.**

**Introduction**



# 1. Thesis Overview

One of the important contributions of medical imaging as the eye of medicine is not only to provide the ability to look inside the patient non-invasively, but also to fuse various available information sources by establishing spatial or temporal correspondences, and therefore fully exploiting the often complementary information.

Various imaging modalities nowadays provide insight into the patient's body without the need to perform surgery, examples include ultrasound, CT, SPECT, MR, etc. One of the oldest examples is X-Ray imaging as invented by Wilhelm Röntgen. Planar X-Ray imaging only displays the accumulated information along the path the X-ray follows resulting in a 2D image without depth information. With computationally assisted tomography, 3D volumes can be reconstructed providing true insight into anatomical structures and functioning.

Tomographic reconstruction is the process of converting a series of observations of some effect, often 2D projection images, into information on physical properties inside an object, for example attenuation coefficients. The reconstruction process provides the information that medical imaging can then display for diagnostic purposes or interventional guidance.

Different modalities provide different information. CT or MR volumes provide insight into anatomical structures; functional imaging like SPECT or PET provides insight into metabolic processes, flow processes or other functions of the body. This information is most useful if it is collocated with anatomical information. The process of establishing this collocation is called registration.

This thesis deals with reconstruction and registration in various contexts. Especially, the acceleration of reconstruction and registration are treated. When trying to accelerate algorithms, one choice is throwing more hardware at the problem, the other choice is the development of more efficient algorithms. While we could rely on the processor industry to provide faster and faster processors automatically with each generation in the last decades, this trend is coming to an end. At the same time, massively parallel hardware architectures are available as main stream graphics cards with impressive computational throughput. This processing power comes at a price though, instead of a convenient sequential processing model, these architectures require the reengineering of the algorithms, sometimes new algorithms to really exploit the available capabilities. Luckily, many of the reconstruction and registration algorithms exhibit an innate parallel structure that makes this possible. This approach is used both for registration and reconstruction in this thesis.

The other option is the development of novel algorithms that run more efficiently or use additional knowledge in the form of previously acquired input or information on the structure of the problem. Using additional knowledge is used both for registration as well as reconstruction in this thesis.

The first contribution of the thesis deals with the GPU-accelerated registration of deformable multi-modal volumes. This part develops algorithms for registration that exploit graphics cards in order to accelerate the algorithm. At the same time, this algorithm includes prior knowledge in the form of statistical intensity priors. Likewise, the reconstruction of SPECT volumes is accelerated using GPUs.

The second option of using newly developed algorithms is then used to update reconstructed functional volumes intra-operatively. This approach uses previously acquired knowledge in the form of a volume that provides information on the number and approximate location of the structures and then performs a registration to intra-operatively acquired probe readings which in effect results in a reconstruction.

The following text provides a brief overview of the structure of the thesis.

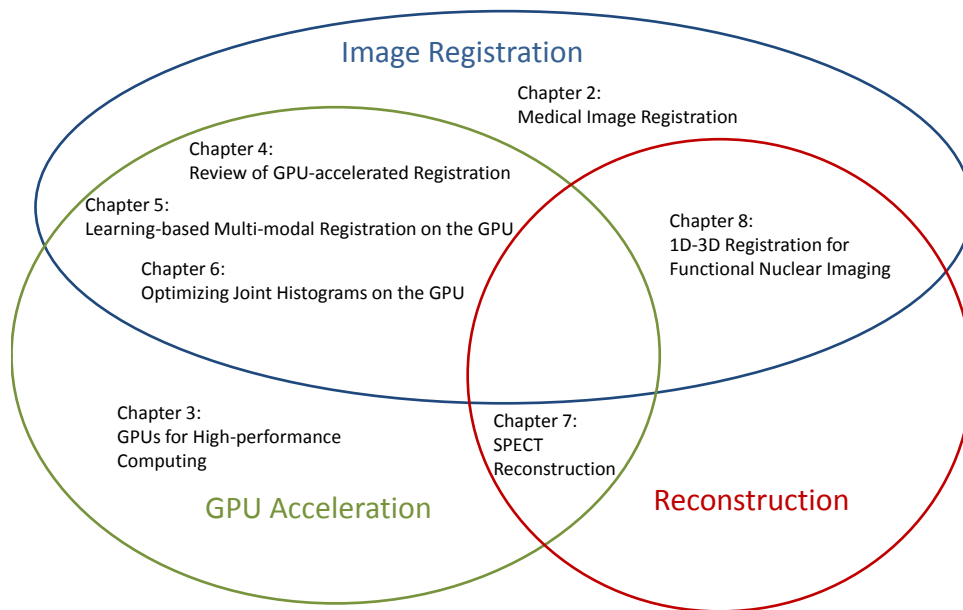


Figure 1.1.: This image gives a graphical overview of the different areas (image registration, GPU acceleration and reconstruction) that are touched by the different chapters in the thesis.

## Medical Image Registration

The basics of medical image registration are introduced in this chapter including the various components, like interpolation, transformation, regularization, similarity measures and optimization. This chapter lays the foundation for the more specialized discussion of registration in different application contexts in the later chapters.

## GPUs for High-Performance Computing

The GPU programming model is introduced as a basis for GPU-accelerated algorithms in the following chapter. This starts with an overview over the graphics pipeline as the motivating factor for the development of graphics processing units. The evolution of graphics hardware is traced with its evolving capabilities and contrasted with the more familiar CPU programming model. Finally, CUDA is introduced as an example for a language exposing the massively parallel hardware architecture of graphics processing units for general purpose applications.

## Review of GPU-Accelerated Registration

Based on the concepts that we have introduced in the previous chapter, we give a survey over how GPUs have been applied to accelerate image registration. We discuss the advantages of the GPU programming model for image registration as well as the inherent limitations, and furthermore describe



---

the details of the important building blocks for successful implementations. This chapter is based on our survey on GPU registration from 2010 [1].

## **Learning-based Multi-modal Registration on the GPU**

Harnessing the computational power of graphics processing unit, a learning-based multi-modal deformable registration method is accelerated. The complete registration pipeline is implemented on GPU hardware. This chapter details the application of the GPU programming model to this specific registration algorithm. A clinical use case is introduced as motivating example, then the mathematical basis of the algorithm is detailed, followed by an in-depth explanation of its technical implementation. This chapter provides both a first example of GPU-acceleration of algorithms as well as an example of leveraging prior knowledge in the form of statistical intensity priors. This chapter is based on two papers on GPU-accelerated registration [2, 3].

## **Optimizing Joint Histograms on the GPU**

In this chapter, one of the components of the registration module is examined in more detail. Despite the impressive computational power of GPUs, one of the most-used similarity measures for multi-modal registration - mutual information - is not well-suited for the streaming architecture because of its memory access pattern. By careful investigation of the performance bottlenecks, the hardware implementation is tuned using the latest GPU capabilities. This part demonstrates how apparently simple sequential algorithms require a detailed examination as well as much more complicated software in order to run efficiently on massively parallel hardware. Two optimization approaches improve the performance by a factor of 4 compared to state-of-the-art GPU algorithms in the latest research papers. This chapter is based on the following paper: [4].

## **SPECT Reconstruction**

Another key process in imaging procedures, before registration and visualization can be even performed, is the reconstruction of the volumes from the scanner output. This chapter introduces the basic foundations of tomographic reconstruction geared towards the reconstruction of SPECT volumes. The physics of SPECT imaging are explained as well as the principles of reconstructing SPECT images including the challenges and problems unique to this modality. We present a GPU-accelerated implementation of single photon emission computed tomography (SPECT) reconstruction based on an ordered-subset expectation maximization algorithm. The algorithm uses models for the point-spread-function (PSF) to improve spatial resolution in the reconstruction images. The hardware-accelerated algorithm is presented in detail (based on [5]).

## **1D-3D Registration for Functional Nuclear Imaging**

This chapter builds on the physics of SPECT reconstruction and the basics of registration introduced earlier in order to perform an effective reconstruction using registration (based on [6, 7]). 3D functional nuclear imaging modalities like SPECT or PET enable surgeons to label small structures with radioactive tracers and locate them precisely. This positional information is valuable intra-operatively as well, for example when locating potentially cancerous lymph nodes in the case of breast cancer. However, the volumetric information provided by pre-operative SPECT scans loses validity quickly due to posture changes and manipulation of the soft tissue during surgery. During the intervention, the surgeon has to rely on the acoustic feedback provided by hand-held gamma-detectors in order to localize the marked structures precisely.

We present a method that allows updating the pre-operative image guidance with a very limited number of tracked detector readings. A previously acquired 3D functional volume serves as prior knowledge and a limited number of new 1D measurements is used in order to update the prior knowledge. This update is performed by a 1D-3D registration algorithm that registers the volume to the detector readings. This enables the rapid update of the visual guidance provided to the surgeon during a radio-guided surgery without slowing down the surgical workflow. We evaluate the performance of this approach using 2Monte-Carlo simulations, phantom experiments and patient data, resulting in a positional error of less than 8 mm. The 1D-3D registration is also compared to a volumetric reconstruction using the tracked detector measurements without taking prior information into account, and achieves a comparable accuracy with significantly less measurements.

**Part II.**

**State of the Art**



## 2. Medical Image Registration

This chapter introduces the principles of medical image registration ([8]), the mathematical foundations as well as the notation that is used in the following chapters for more specific applications. Medical imaging allows the physician to peek inside the body of the patient and observe the size, position, structure and relationship of internal organs as well as spatial information about pathologies or functions in the human body. Image registration helps the physician make sense of these images by aligning images so that corresponding features can be easily related.

Medical image registration establishes spatial (or possibly temporal) relationships between images of different modalities, depicting anatomical, functional, physiological or pathological information. Examples for different image sources are CT [9] or MR scans [10] that provide anatomical information, with CT especially providing information about bone structures, whereas MR provides detailed images of soft tissue. PET [11] or SPECT scans [12] use radioactive markers and provide insight into functional processes in the body depending on the kind of marker that is used for the imaging. Establishing spatial relationships between these different images enables inter-patient and intra-patient studies. Fusing different modalities provides additional information not apparent from one modality alone. Image guided interventions fuse pre-operative data almost routinely, for example anatomical information from a CT scan, with data from imaging modalities that are more amenable to interactive use during a surgery [13, 14].

Image registration establishes the spatial relationship between these images via common attributes. Since the transformation parameters from one image to the other are inferred from the images, image registration is classified as an inverse problem [16].

Registration is a well-researched area with a multitude of published literature. Surveys of the field can be found for example in [17]. For a survey of image registration not geared towards medical images in particular see [18].

### 2.1. Components of Image Registration

When registering two images, one is usually considered the fixed image (also called source image or reference image)  $I_f(X)$ , the second image is the moving image (also called the target image or template image),  $I_m(\Phi(X))$ , with  $I : \mathbb{R}^n \rightarrow \Omega$ . While the fixed image remains unchanged, the moving image is transformed by  $\Phi$ . This is performed iteratively in order to find the transformation  $\hat{\Phi}$  that maximizes the similarity function  $S$  between the two images. Registration is then the solution to the following equation:

$$\hat{\Phi} = \arg \max S(I_f(X), I_m(\Phi(X))) \quad (2.1)$$

The goal of this procedure is to map every point in the fixed image to correct corresponding point in the moving image. This correct point depends on the process that leads to the differences between the fixed and the moving images. When registering two images of the same patient, several factors (for example different body positioning, breathing motion, growth in children, or physiological changes) lead to differences so that determining corresponding points depends on that information that is not necessarily available to the algorithm. When performing intra-subject registration for example for the generation of an atlas, the problem of defining the exact corresponding point becomes even more difficult. Due to these difficulties a wide range of measures have been researched that measure the similarity between images. These measures guide the optimization process that determines optimal parameters for the transformation between two images according to the similarity measure used. This optimization process is usually an iterative process.

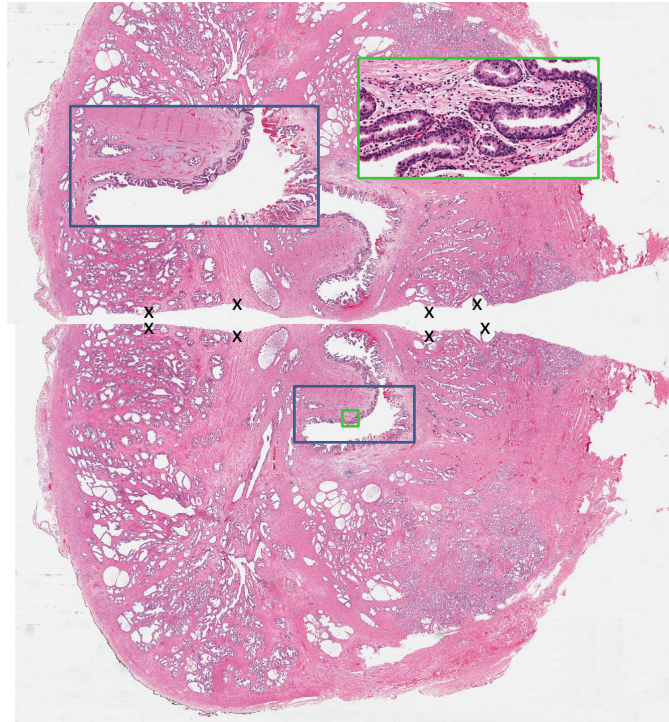


Figure 2.1.: One example for medical image registration are prostate histology images that often have to be split into different quadrants due to technical limitations of the slicing apparatus. For further processing, these slices have to be stitched together again. Corresponding landmarks in adjacent quadrants are selected, so that the registration algorithm can align the quadrants as shown in our paper from 2012 [15] .

The components of a registration system are the optimization procedure, the transformation, the interpolation function, the similarity measure  $S$ , a regularization procedure and projection operators  $P_f$  and  $P_m$ . The implementation and types of these components depend on the registration problem at hand. Not all registration procedures require all the components.

The optimization part performs the search for the optimal transformation parameters. The transformation type defines the allowed search space for the optimization, for example whether the images can only be transformed rigidly or deformed according to some rules as well. The interpolation function allows treating the images as continuous, even though they are saved as discrete samples in the computer. The similarity measure guides the optimization algorithm towards a transformation that best matches corresponding points in the two input images. Projection operators are required when images of different dimensionalities are to be registered for example 2D X-Ray images with 3D CT scans.

$$\hat{\Phi} = \arg \max S(\mathcal{P}_f(I_f(X)), \mathcal{P}_m(I_m(\Phi(X)))) . \quad (2.2)$$

In the following, we will discuss the different components of image registration in turn.

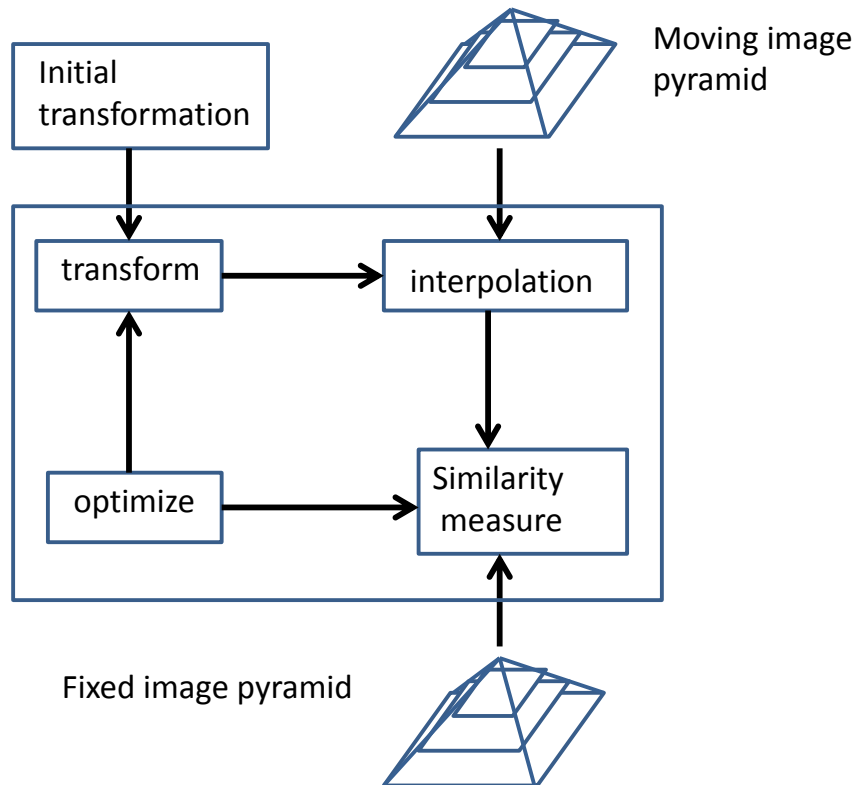


Figure 2.2.: registration overview

## 2.2. Image Representation

Before we turn to the different components of the equation 2.2, we have to model the images that the registration algorithm operates on. For our purposes, images can be modeled as scalar- or vector-valued functions on vectors, with the image  $i$  a function  $i : \mathbb{R}^n \leftarrow \mathbb{R}^m$  [19]. In the case of modalities like CT, SPECT, or MR  $m = 1$ , since only one intensity value is recorded at each position. For color images, three color channels are sampled, so  $m = 3$ . One example for this are the histology slices in figure 2.1. Registration problems can also be divided according to the dimensions of the images that are registered. Common examples are 2D-2D registration in the case of image to image registration, or 3D-3D registration for medical volumes. Another common use case is 2D-3D registration. One example for this is the registration of an intra-operative X-ray to an pre-operative CT scan, in order to provide the surgeon with the additional information from the CT scan and to give more 3D spatial information. Another example in the later chapters is the introduction of 1D-3D registration.

## 2.3. Interpolation

Treating images as continuous functions, as we have done in the previous section, is very convenient, but the images provided are of course discretely sampled  $I : \mathbb{Z}^n \rightarrow \mathbb{Z}^m$ , since images are discretized in the computer [20]. Therefore, only samples at specific locations are available, often at the position of a rectangular grid. Since corresponding points in the two images do not necessarily coincide with grid points and the images have in general different resolution, we require interpolation schemes that

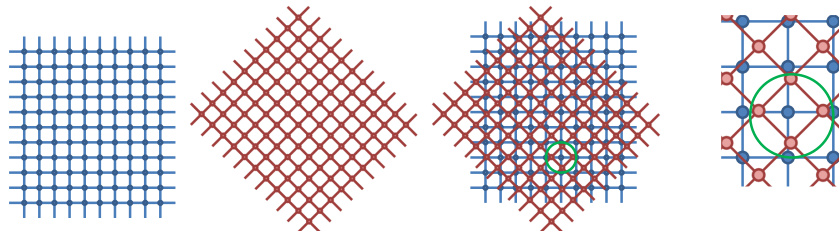


Figure 2.3.: This diagram visualizes the need for interpolation functions. The two images are sampled at the grid positions of the red and the blue grid. When a rotation is performed, the grid positions do not coincide anymore. When evaluating a similarity measure for example for the blue grid position inside the green circle, the interpolation function takes surrounding grid positions of the red grid into account, in order to compute an intensity value at a corresponding position.

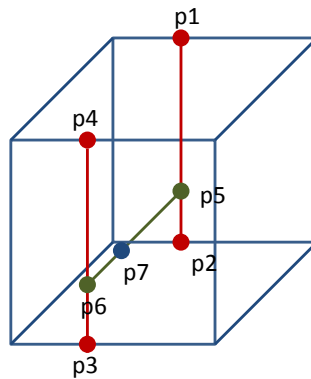


Figure 2.4.: Trilinear interpolation performs linear interpolation resulting in the 4 red sample points. Linear interpolation again, produces 2 bilinear samples in green, and a final linear interpolation results in the trilinear sample in blue.

provide data intensities at all spatial positions. Interpolation functions compute the intensity values at arbitrary locations that do not coincide with grid positions and allow us to then treat images as continuous. Figure 2.3 provides a diagram of two images that are rotated against each other and therefore cannot be compared at exact grid locations.

These interpolation functions  $L$  perform the interpolation between grid points and allow us to assign values to every point within the image.  $L$  is defined as with  $L : \mathbb{Z}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ . The parameters of this function are a  $d$ -dimensional image and a position vector of the same length and the output is the interpolated intensity at this point.

The appropriate choice of interpolation schemes is important since interpolation errors can significantly impact the registration accuracy [21, 22]. The simplest interpolation scheme is nearest neighbor interpolation that just chooses the nearest grid point. While it is computationally very efficient, it soon leads to artifacts.  $D$ -linear interpolation (in the case of volumes 3D interpolation) is a scheme that is often chosen because it provides a good trade-off between speed and accuracy. A graphical representation of this interpolation scheme can be seen in figure 2.4. Other common options are



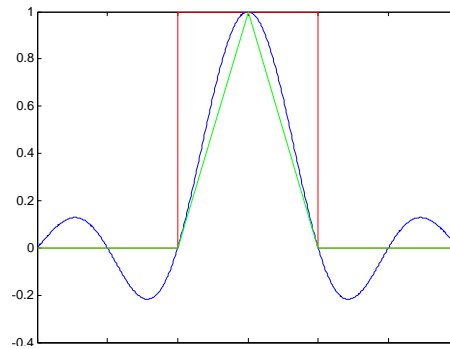


Figure 2.5.: This plot visualizes three interpolation kernel functions for the 1D case. The nearest neighbor interpolation function is plotted in red, the linear interpolation function in green and the sinc function in blue.

quadratic interpolation functions or B-splines. The theoretically optimal interpolation function is represented by the sinc function (also displayed in figure 2.5, see [23] for more detail).

$$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x} \quad (2.3)$$

The sinc function corresponds to a box filter in the frequency domain. However, this interpolation function requires infinite support, but truncated implementations are possible. Next to these interpolation schemes, a wide variety of interpolation schemes with various trade-offs have been proposed for computer graphics and image processing. For a more comprehensive survey, see Lehmann et al. [24].

## 2.4. Transformation

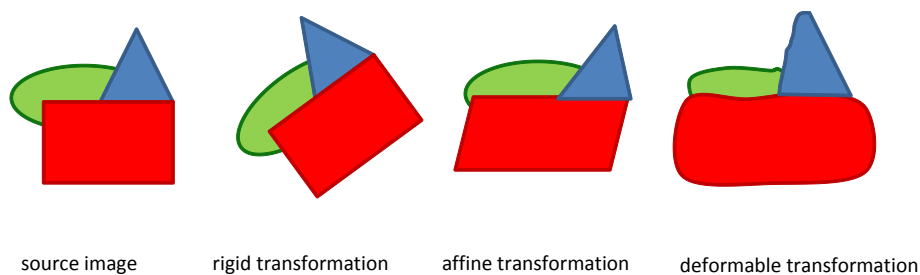


Figure 2.6.: This figure visualizes common transformation types. The left side shows the original image, a rigid transformation performs translations and rotations, an affine registration might include shear, and the deformable registration is unrestricted in which deformations it might perform.

Now that interpolation allows us to treat images as continuous functions, we can use transformations to map the moving image onto the fixed image. The registration procedure aims to find the best mapping that aligns the two images. Figure 2.6 shows an overview of commonly encountered transformation types for image registration. Transformations can be rigid, affine or deformable.

### 2.4.1. Affine Transformations

The simplest type of transformation is the rigid transformation that only applies translation and rotation to the moving image. This type of transformation is often appropriate for relatively rigid structures, for example the human skull. Rigid transformations do not change geometric properties like angles between structures or the length of distance. The rigid transformation  $\Phi$  can be represented as a rotation matrix and a translation vector, or - as is common in computer graphics - using a single  $4 \times 4$  matrix with homogeneous coordinates [25].

$$\Phi(x) = Rx + t = Ax \tag{2.4}$$

with

$$A = \begin{pmatrix} r_{00} & r_{01} & r_{02} & t_0 \\ r_{10} & r_{11} & r_{12} & t_1 \\ r_{20} & r_{21} & r_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.5}$$

The Matrix  $R$  is orthonormal with  $\det(R) = 1$ . For optimization, the coefficients of the rotation matrix are therefore not manipulated directly, but a different parameterization is chosen that is used to derive the rotation matrix. A common choice for parameterization are Euler angles. Euler angles have the disadvantage of a possible loss of degrees of freedom (also called a gimbal lock), but this does not pose a problem if the angles of rotation are rather small, as is common in image registration. Another common parameterization uses quaternions for representing rotations.

If additional degrees of freedom in the form of shear and scaling are added to the transformation, the number of degrees of freedom is increased from 6 to 12, since the coefficients of the matrix  $A$  are not restricted anymore, as was the case for rotation. These transformation functions are called affine transformations and are described by an affine matrix  $A$ . As is the case for rigid transformations, the coefficients of the matrix  $A$  are not manipulated directly, but instead, 3 parameters for translation, rotation, shear along each axis and scaling along each axis are manipulated and the affine matrix is then derived from these parameters in order to perform that actual transformation.

$$\Phi(x) = Ax \tag{2.6}$$

The much smaller number of parameters of the affine transformation enables an easier optimization. Furthermore, the affine and especially the rigid transformation model is intuitive to the end user and therefore widely used in clinical practice. However, affine transformations are restricted and might not be able to approximate typical deformations in medical image registration for example those caused by breathing motion.

### Projection

Projection is needed when images of different dimensionalities need to be registered. 2D-3D registration problems are quite common [26, 27, 28], but 1D-3D registration problems exist as well (see [6]). 3D scanning modalities often have a longer acquisition time, whereas many 2D scanning techniques are fast and can be performed during the intervention as well [13, 14]. Minimally invasive procedures, computer-aided surgery and radiation therapy all benefit from instantly updated imaging information. This imaging information can then be registered with 3D information that has been previously acquired for planning or diagnostic purposes.

With respect to Eq. 2.2 a projection of a higher-dimensional image (n-D image) to a lower-dimensional image (m-D image) corresponds to  $\mathcal{P}_h$ , and  $\mathcal{P}_f$  equals the identity transformation. Thus,  $\mathcal{P}_f : \mathbb{R}^m \rightarrow \mathbb{R}^m$  and  $\mathcal{P}_h : \mathbb{R}^n \rightarrow \mathbb{R}^m$ .

$\mathcal{P}_m$  performs the projection of the n-dimensional image onto the m-dimensional image.

Projection operators are not only useful when the dimensionality of the registered images is different but also in order to reduce the computational complexity. See [29] for an example that uses projected 2D images to register two 3D volumes. Difficulties for projection-based registration that occur due to

dissimilarities between the intra- and pre-operative imaging data due to the presence of contrast agents or surgical instruments in the field of view are for example addressed in [30].

## DRR

The registration of 3D CT volumes to 2D X-Ray images is especially common. It is also a special case, since X-Ray images can be simulated from CT volumes, because CT uses X-Rays as well. The X-Ray images that are simulated from the CT volumes are called digitally reconstructed radiographs (DRR) [31]. The computation of a DRR is performed by evaluating the ray integral according to the following formula

$$I(x) = I_0 e^{\left(-\int_{r(x)} \mu(X, E_{\text{eff}}) dr\right)}, \quad (2.7)$$

with  $x \in \mathbb{R}^2$  and  $X \in \mathbb{R}^3$  and the attenuation factor  $\mu(X, E_{\text{eff}})$ . The final intensity  $I(x)$  in the image at position  $x$  is the energy of the photons that reach the detector after they have been attenuated on the path from the photon emitter by the various tissues in the patient's body. While DRR can be used with projective transformations, they might also be used with more complex deformable transformations, as introduced in the following section.

### 2.4.2. Deformable Transformations

#### Dense Vector Fields for Deformable Transformations

Affine transformation are not necessarily general enough for registration, since they cannot capture the behavior of soft tissue or even the effect of breathing motions on the ribcage. Deformable registration [19] enables much more flexible deformations, but at the cost of an equally much higher number of parameters to optimize. A very general representation for deformable registration consists in defining a dense field of displacement vectors  $u(x)$ . Such a field defines a vector for every possible location  $x$ . The vector field can be defined analytically or by storing the displacement vectors directly. In this case, the vectors have to be interpolated similar to the interpolation of intensity values of the image itself.

$$\Phi(x) = x + u(x) \quad (2.8)$$

with  $u(x)$  a displacement vector.

#### Control Point Schemes for Deformable Transformations

Instead of using a dense vector field in order to model the deformation, using control-point based methods are also a popular choice for registration models. Free-form Deformation (FFD) [32] is a way to define a dense vector field that is determined by a low number of control points. This reduces the computational load and includes some regularization as well by restricting the solution space. Instead of solving for a number  $n_x n_y n_z$  of vectors only  $m_x m_y m_z$  control points  $\phi_{i,j,k}$  with initial spacing  $\delta$  need to be found. The displacement vectors are then computed by interpolation functions depending on the control points

$$\Phi(X) = \sum_{l=0}^d \sum_{m=0}^d \sum_{n=0}^d B_l(u) B_m(v) B_n(w) \phi_{i+l, j+m, k+n}, \quad (2.9)$$

with degree  $d$  and basis function  $B$ .

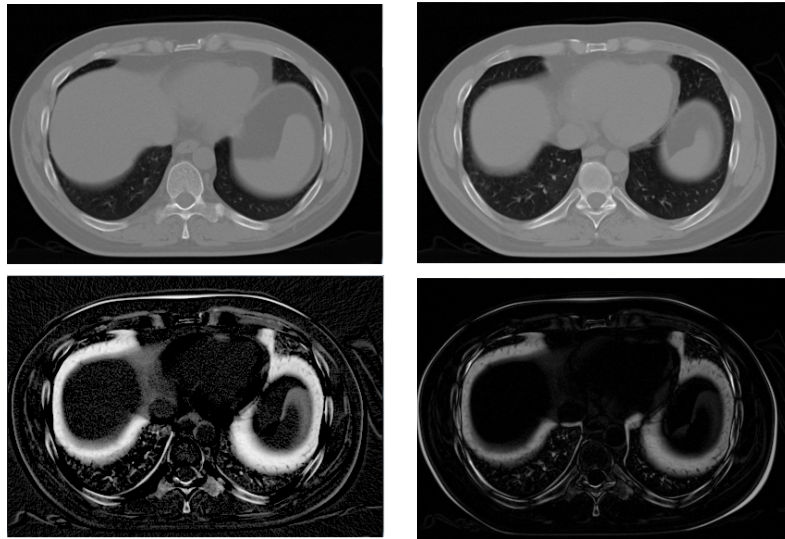


Figure 2.7.: An example for a registration problem is the registration of two slices of a CT volume as can be seen in this figure. Below, the visualization of the error metrics SSD and SAD is depicted.

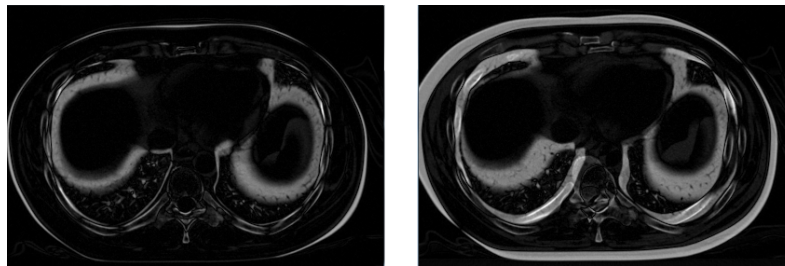


Figure 2.8.: If the two CT slices from the example above are offset against each other, the error metric (in this case SAD) shows higher error values especially at the edges in the image.

## 2.5. Similarity measures

Similarity measures give a metric for the similarity between two images and guide the optimization procedure with this. The similarity measure depends on the type of images that are to be registered. A good similarity measure has its global optimum at the point that represents the correct alignment of the moving and the fixed image. An additional desirable feature is a smooth increase when moving from parameters that are somewhat wrong, to parameters that are closer to the optimum. Robustness to noise and features that might only be present in one of the images (for example surgical instruments) is another important feature. Similarity measures can be represented mathematically as operators that take two images and a transformation as input and return a value that describes how well the images are aligned. The similarity measure has an optimum, when the two inputs are aligned. Ideally, the return value decreases smoothly without additional local optima when the images become gradually unaligned. Instead of similarity measures, error metrics can be used that measure the error

in the alignment of two images. Minimizing the error in the alignment of the two images is equivalent to maximizing the similarity between the two images.

Similarity measures fall into two broad categories:

- landmark-based
- intensity-based.

Landmark-based registration [20] is an intuitive approach in which corresponding landmarks between two images are determined. For rigid structures, the non-collinear points are sufficient, however, usually more points are used to make the procedure more robust against errors. For non-rigid structures, more landmarks have to be used. The landmarks can either be set manually or computed in a preprocessing step.

Measures that work on the intensities of the voxels in the volume directly do not need such a preprocessing step. Common similarity measures are sum of squared distances (SSD), sum of absolute distances (SAD), mutual information (MI), cross correlation (CC), or normalized cross correlation (NCC).

In the following, we concentrate on some commonly used voxel-based similarity measures, as they are more relevant for the following chapters.

### 2.5.1. Sum of Squared Distances

The Sum of Squared Distance (SSD) [8] is based on the assumption that corresponding points in the image have the same intensities. It is one of the simplest similarity measures and is computed as:

$$I_{SSD}(U) = \frac{1}{|I_{f,m}|} \sum_{I_{f,m}} (i_f - i_m)^2 \quad (2.10)$$

$I_{f,m}$  in the equation is the overlap of the two images  $I_f$  and  $I_m$ . This measure is appropriate for images of the same modality. It can be shown that SSD is the optimal measure if the difference between the two images is only Gaussian noise. This is not the case in inter-modal registration and in most cases neither in intra-modal registration. In medical registration, structures might be deformed, contrast agents have been injected or surgical instruments might be visible. The SSD is sensitive to extreme changes in voxel intensity that are introduced by these changes, even if the number of voxels is small. This problem especially appears in the case of interventional imaging, because surgical instruments might be visible at different positions and unduly influence the registration result. Another source of problems is the presence of contrast agent that changes the pixel intensity drastically. Another problem might arise from the normalization procedure. The SSD measure is divided by the number of overlapping pixels. If that normalization is not performed, an optimum might be found by just overlapping very few similar pixels, resulting in a wrong transformation. On the other hand, if the two images contain lots of noise and therefore only have a weak optimum at the point of the best alignment, a full overlap could produce an optimum, because the denominator becomes very small in the case of a full overlap.

### 2.5.2. Sum of Absolute Distances

Another equally simple similarity measure is the Sum of Absolute Distances (SAD) [8]:

$$I_{SAD}(U) = \frac{1}{|I_{f,m}|} \sum_{I_{f,m}} |i_f - i_m| \quad (2.11)$$

Instead of the squared difference between the intensities, this measure uses the absolute difference between the intensities. This gives less weight to extreme outliers, so it reduces the sensitivity of this measure to extreme intensity changes in a small number of voxels. But it is still based on the same assumption of constant intensities, so SAD often has similar problems to SSD, though they are less pronounced.

### 2.5.3. Normalized Cross Correlation

If one cannot assume constant intensities, as is necessary for SSD and SAD, but the weaker assumption holds, that there is a linear relationship between pixel values, the Cross Correlation (CC, often also called correlation coefficient), or the Normalized Cross Correlation (NCC) [8] are appropriate measures. The NCC is a common algorithm used in signal processing and other fields. The NCC is not affected by variations in pixel intensity and contrast, however, regions that have inverse intensity relationships can compensate for each other. Therefore, this measure is not suitable for all kinds of multi-modal registration.

$$I_{NCC}(U) = \frac{\sum_{I_{f,m}} (i_f - \bar{i}_f)(i_m - \bar{i}_m)}{(\sum_{I_{f,m}} (i_f - \bar{i}_f)^2 \sum_{I_{f,m}} (i_m - \bar{i}_m)^2)^{\frac{1}{2}}} \quad (2.12)$$

The CC can be - same as mutual information measure - derived using maximum likelihood see [33].

### 2.5.4. Mutual Information

But even the assumption of a linear relationship between voxel values as needed for the CC might not hold. One example are inter-modal registrations between CT and SPECT. A different approach is needed. This approach is provided by information theoretic similarity measures and has been introduced by two research groups in parallel [34, 35, 36, 37]. The idea behind this approach is to maximize the shared information between two images, or to reduce the information in the combined images. Corresponding structures in the images overlap, when the images are aligned correctly. This is interpreted as shared information. Entropy is used to measure the information (entropy has also been used directly as a similarity measure, see [38]). Joint entropy correspondingly measures the information in the combined images. The better the alignment is, the lower is the joint entropy.

The joint probability distribution is the likelihood of intensity combinations in the two images. In the discrete case, the joint probability can be approximated by the normalized joint histogram. The joint histogram counts the occurrence of each intensity combination in the two images. These counts are divided by the number of pixels in the overlap region for the normalization.

The MI (mutual information) similarity measure is computed as:

$$I_{MI}(U) = - \int_{I_f} \int_{I_m} p_U(i_f, i_m) \log \frac{p_U(i_f, i_m)}{p(i_f)p_U(i_m)} di_f di_m, \quad (2.13)$$

with  $U$  denoting the transformation that aligns the fixed image  $I_f$  with the moving image  $I_m$ , and  $i_f$  and  $i_m$  the intensities on the fixed respectively moving image. The letter  $p$  denotes probabilities, with  $p(i_f)$  the probability of intensity  $i_f$  in the fixed image and with  $p_U(i_m)$  the probability of the intensity  $i_m$  in the moving image.  $p_U(i_f, i_m)$  is the probability that intensity  $i_f$  and  $i_m$  occur at corresponding points in the two images given the transformation  $U$ . The integration is performed over the number of bins  $m \times f$  of the fixed and moving image. The joint histogram stores for every image intensity in the fixed image, how often this intensity coincides with a specific intensity in the moving image. For a derivation based on maximum likelihood see [39].

The original mutual information measure depends on the overlap area which might result in misregistration. Several normalization schemes [8, 40] aim to solve this problem. The mutual information measure is a stable similarity measure for mono-modal and multi-modal registration, however, it does not take spatial information into account but relies on statistical information exclusively. Since it relies on a larger amount of voxels and does not assume a linear relationship, it does not excel in cases where the number of voxels is small or a linearity can be assumed as is the case for many 2D-3 registration applications (for example the registration of a CT to a fluoroscopy image [41]). For a more thorough review of the literature regarding registration based on mutual information see [42].

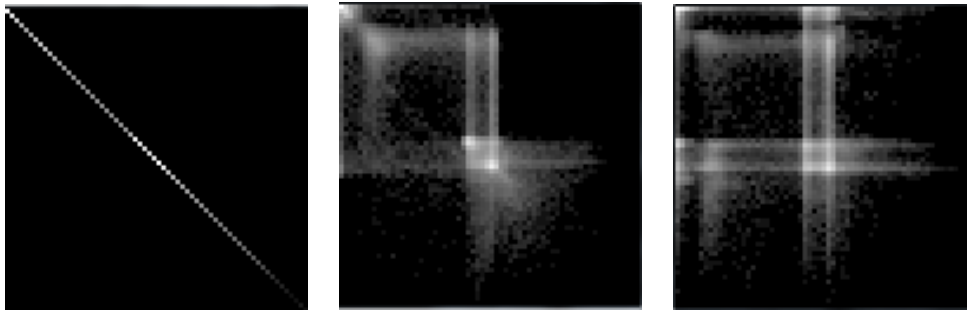


Figure 2.9.: The figure on the left shows the joint histogram for two perfectly matching (identical) images, the middle shows an example of two aligned slices from slightly different horizontal positions, and the right shows the joint histogram for the same two slices with an additional offset between matching positions.

### 2.5.5. Kullback-Leibler Divergence

Instead of minimizing the shared information between different images, using the Kullback-Leibler divergence [43, 44] allows one to add prior information using a previously acquired histogram (and this complements the MI measure as we will see in chapter 5). This histogram encodes a-priori knowledge of the mapping between the pixel intensity in the moving and the fixed image. The input histogram can be derived from registration where the correct transformation is known or by using expert-registered datasets. The distance between the two histograms is estimated using the Kullback-Leibler divergence and results in the following equation:

$$I_{KL}(U) = - \int_{I_f} \int_{I_m} p_U(i_f, i_m) \log \frac{p_U(i_f, i_m)}{p^f(i_f, i_m)} di_f di_m, \quad (2.14)$$

Same as the mutual information similarity measure, the Kullback-Leibler measure depends on statistical information and does not take spatial information into account. It also relies on the a-priori knowledge to be accurate.

### 2.5.6. Other Similarity Measures

Besides the similarity measures mentioned above, there exists a wide variety of other, less commonly used similarity measures, proposed in the literature that are tailored to specific applications. We will just mention a few here, like the ratio of image uniformity (RIU) [45], pattern intensity (PI) [46], correlation ratio (CR) [47, 48, 49] used for 2D-3D registration [50], gradient correlation (GC) or gradient difference (GD) [41], sum of local normalized correlation (SLNC) and variance-weighted sum of local normalized correlation (VWC) [51], or joint gradient [52]. Especially for 2D-3D registration, similarity measures based on the gradient image are commonly used ([53]). For an overview of different similarity measures see also table 2.2

For a more thorough discussion of various similarity measures see [8] and their advantages and disadvantages [54, 55, 41, 56].

## 2.6. Optimization Procedure

With the similarity measure in place, the registration process has to find the transformation that results in the best similarity. The optimizer is the part of the registration pipeline that maximizes the similarity measure (see equation 2.1). Evaluating the similarity measure is usually a costly operation

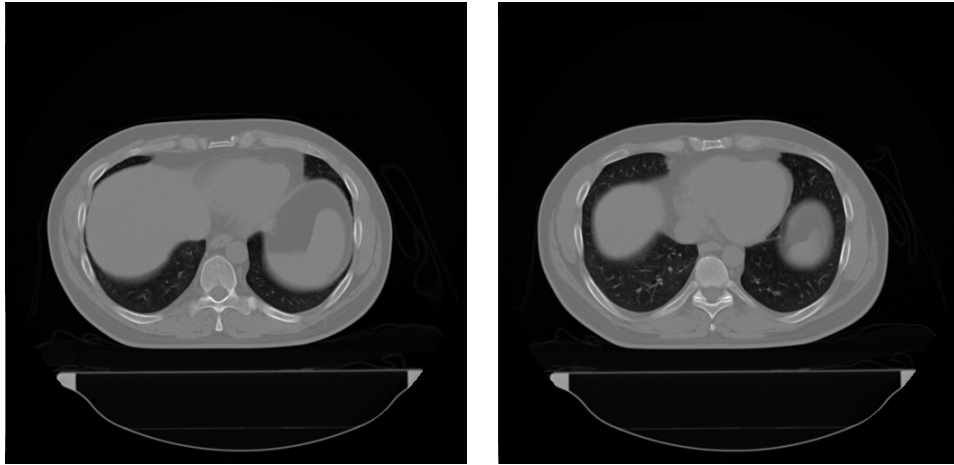


Figure 2.10.: This image shows two slices of a volume that are used for the plotting of the behavior of the similarity measure in figure 2.11. The slices are shifted diagonally against each other and different similarity measures are evaluated.

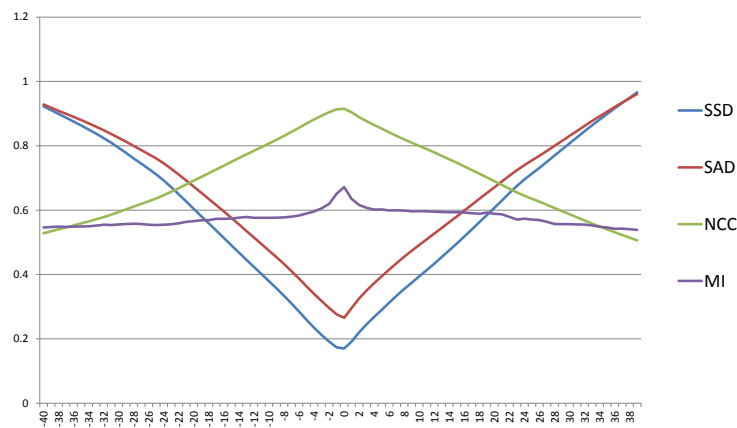


Figure 2.11.: This figure shows an evaluation of the behavior of different similarity measures for the slices in figure 2.10. This graph shows the SSD (sum of square distances), SAD (sum of absolute distances), NCC (normalized cross correlation) and MI (mutual information). The shift on the x-axis is in voxels for this experiment. The SSD and SAD values have been scaled to fit into the same graph as NCC and MI.

for image-based registration, since quite large images have to be transformed and evaluated at every grid location. An exhaustive search, even for a relatively small number of free parameters is prohibitively expensive. Therefore, different optimization strategies have been developed that try to minimize the evaluation of the similarity measure and try to determine the next evaluation location intelligently [60]. These evaluations are performed iteratively until the stopping criteria are met, which can be a maximum number of iterations or similarity measure evaluations. Another common stopping



Table 2.1.: Reported Implementations of Common Similarity Measures and their Characteristics

Measure	Advantages	Disadvantages
SSD	Simplicity, high speed	Sensitive to outliers not for different modalities.
SAD	Simplicity, high speed	Same problems as with SSD but less sensitive to outliers.
NCC	Unaffected by variations in contrast and brightness.	Regions with inverse intensity relation can have a compensating effect. Not necessarily suitable for multi-modal registration.
MI	Stable measure for multi-modal registration.	Relies exclusively on statistical information of available inten- sities. No spatial information is considered.

criterion is met when the change in the similarity measure between consecutive iterations falls below a predefined threshold, or the changes in the free parameters are smaller than a predefined parameter threshold.

Optimization can be broadly divided into methods that descend along the gradient of a cost function (gradient-based) and methods that do not depend on the gradient (gradient-free). While gradient-based methods usually provide a faster convergence towards the optimum, they are also more complicated since they either require a closed-form solution or a numerical approximation of the gradient.

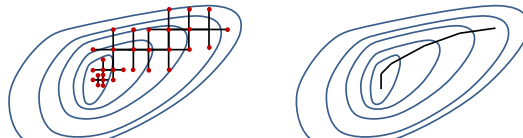


Figure 2.12.: This figure shows the two common optimization approaches. On the left side the process of optimization using best neighbor search is depicted. The red dots represent evaluations of the cost function. Starting from the initial position on the right side, this algorithm simply evaluates neighbors in the parameter space and selects the best parameter as the next estimate. If there are no better neighbors, the step size is decreased in order to refine the optimal parameter.

The type of transformation is an important aspect since it determines the number of free parameters that need to be optimized. In the case of a rigid transformation (appropriate for registering the human skull for example), there are only 6 free parameters, 3 parameters for the translation and another 3 that describe the rotation. In the case of general affine transformations that can be described by a 4x4 matrix, 12 free parameters are available (translation, rotation, shear and scale). For a non-affine, deformable registration on the other hand there might be as many parameters as there are voxels within the image. With only a limited amount of data a common strategy for gradient-based optimization is to compute the cost function twice for each parameter and form a gradient vector out of the result. In the following, some common optimization strategies for registration are discussed.

Table 2.2.: Abbreviations for Similarity Measures and References

Acronym	Name	Reference
CC	correlation coefficient	[8]
CR	correlation ratio	[47, 48, 49]
GC	gradient correlation	[41]
GD	gradient difference	[41]
GDGP	gradient proximity	[29]
JG	joint gradient	[52]
KL	Kullback-Leibler	[43, 44]
LNC	local normalized correlation	[57]
MI	mutual information	[34, 35, 36, 37]
NCC	normalized cross correlation	[41]
NMI	normalized mutual information	[8, 40]
PI	pattern intensity	[46]
RIU	ratio of image uniformity	[45]
SAD	sum of absolute distances	[8]
SLNC	sum of local normalized correlation	[51]
SSD	sum of squared distances	[8]
VWC	variance-weighted sum of local normalized correlation	[51]
VOD	variance of differences	[58, 59]

Commonly used optimization procedures are:

- best neighbor search
- simplex method
- Powell-Brent method
- gradient descent optimization.

### 2.6.1. Best-Neighbor Search

Best-Neighbor Search (used for example by Studholme et al. [61]) is one of the simplest optimization strategies available. Starting from an initial position, neighboring locations in the parameter space are evaluated. This is usually done by adding or subtracting a predefined step size from each parameter of the parameter vector in turn. This results in  $2n$  cost function evaluations for  $n$  degrees of freedom. The best neighbor with the best cost function value is then used as the starting point for the next iteration. If there is no improvement in the cost function value, the optimization could either be stopped, or the step size decreased in order to refine the optimum.

While this optimization procedure is easy to implement, it also has the drawback that progressing towards an optimum along narrow ridges can be slow and a step size that is chosen too small might lead to a failure to find the optimum.

### 2.6.2. Simplex Method

The simplex method (also known as the Nelder-Mead method [62, 60]) is another heuristic search method that does not rely on derivatives. A simplex is a minimal geometric shape in  $n$  dimensions with  $n+1$  vertices. The initial simplex around the seed position is evaluated at its corners. Operations like

extension, reflection and contraction are then used to modify the simplex depending on the evaluation result. Compared to the best-neighbor optimization, the simplex method is better at navigating narrow ridges towards an optimum and it uses a smaller number of evaluations for the parameter space. This efficiency comes at the cost of a more complex implementation compared to best-neighbor search.

### 2.6.3. Powell-Brent Method

The Powell-Brent method [63, 60] is another method that does not require the cost function to be differentiable (but requires a smooth function). It works by performing a search along a set of search vectors successively. The new position is then the linear combination of the best positions along the line. After this step, the set of search vectors is adapted. The main complexity lies in the direction search which is performed by Brent line minimization (see chapter 4 in [64]). This root-finding algorithm combines inverse quadratic interpolation, bisection, and the secant method resulting in a algorithm that is fast and reliable, but also more complex to implement than the simplex method.

### 2.6.4. Gradient Descent Optimization

Using the gradient descent (see [65]) is a widely used method for optimization if the derivative of a cost function can be computed, or at least approximated, efficiently. The optimization procedure computes steps proportional to the gradient in order to reach the optimum. The parameter  $\lambda$  is the learning rate that determines how long the steps are. The learning rate has to be tuned to the particular problem. If the learning rate is too small, the algorithm might only find a local optimum, but if the learning rate is too large, the algorithm might skip over the global optimum.

$$x_{j+1} = \lambda \frac{df(x_j)}{d(x_j)} \quad (2.15)$$

Gradient descent is used in the later chapters for optimization (5 and 8).

## 2.7. Regularization

Even with well-chosen optimization algorithms and a fitting similarity measure, registration problems are ill-posed and require a regularization step, in order to restrict the number of possible solutions by providing additional knowledge about admissible solutions. Registration belongs to the case of inverse problems [16], problems that in the most general sense try to convert observed measurements into information about an object. In registration, the observed results are the images and the transformation is the information that is sought.

As typical for inverse problems, registration is one of the problems in image processing that is inherently ill-posed. Even for simple problems, there might not be a unique solution without using additional information (see figure 2.13) and the addition of noise complicates this problem even more. In mathematics, well-posed problems are problems that have the following properties:

1. A solution exists.
2. The solution is unique.
3. The solution depends continuously on the problem data.

If at least one of the conditions does not hold, a problem is called ill-posed. For registration, all three conditions do not necessarily hold: there might not be a solution, there might be several solutions, perhaps several local optima, but possibly several global optima as well and the solution does not depend continuously on the data.

And even if a problem is well-posed, it might still be ill-conditioned, that is small errors in the initial data might lead to large errors in the solution. Deformable image registration can be a highly ill-posed

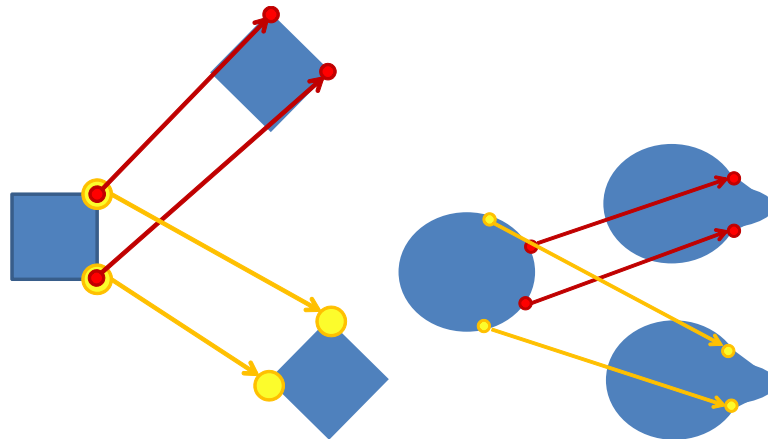


Figure 2.13.: Without additional information, registration problems are often ill-defined. In the diagram on the left, it cannot be decided which points of the original rectangle on the left correspond to which points on the rotated rectangle on the right, since the rotation direction is not known. Another example for the case of deformation is depicted on the right.

problem, as are many inverse problems. The large number of parameters for deformable registration allows arbitrary deformations of voxels that result in an intractably large solution space. Regularization restricts the solution space to make the problem tractable. This reduces the number of local minima, so that the optimization procedure converges more easily to the global optimum. Common approaches to regularization include the following (see also figure 2.14):

1. filtering the resulting deformation fields
2. adding penalty terms to the energy function
3. restricting the transformation space by parameterization of the transformation.

Popular approaches for filtering the deformation field are Gaussian smoothing [66], possibly with different standard deviations for different objects inside the human body [67] or filtering with a rigidity penalty term [68]. Depending on how the filtering is applied exactly, the filtering can be elastic-like or viscous fluid. For elastic-like regularization, the filtering is applied to the total displacement field, whereas for viscous fluid like regularization, the filtering is only applied to the correction field (that is applied to the deformation field so far) of that particular step, since viscous fluid models smooth the velocities of the particles in the fluid.

Instead of filtering the deformation term, penalty terms can be added to the energy functional. Penalty terms can include terms that consider bending energy, linear elasticity, incompressibility, invertibility etc. Instead of using these terms as soft constraints, hard constraints have been added as well [69]. Regularization can also be used implicitly by parameterizing the deformation that is used for the registration process. Examples for parameterizations include free form deformation [70], B-Splines, or Radial Basis functions.

Finally, instead of using functions of space  $L^2$ , using functions from Sobolev spaces [71] offers the advantage that these functions already have regularity properties which is important for deformable registration.

Another problem that regularization tackles for deformable registration particularly is the problem of physical constraints on the resulting deformation field. A completely arbitrary deformation field could result in unphysiological deformation for example foldings, lack of volume preservation, and

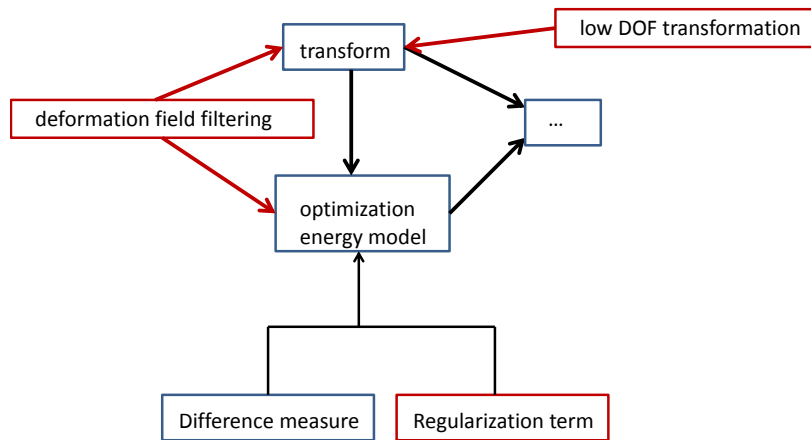


Figure 2.14.: This diagram gives an overview over different regularization strategies and where they might fit into the registration pipeline. Filtering of the deformation field can be part of the transformation or the optimization of the energy model. Using a transformation with a lower degree of freedom for regularization makes the regularization a part of the transformation component. Finally, regularization can be integrated into the energy model.

deformation of bony structure. Simple smoothing already prevents the first problem, but for the latter two, additional physical constraints are necessary, up to a finite element model of the tissue properties.

## 2.8. Conclusion

Registration is a difficult problem, since different transformations and similarity measures need to be chosen for different applications. Correspondence between medical images is correspondence between tissue, but what does this mean for inter-patient registration as just one example? Depending on the processes that lead to differences between images, very different results and very different algorithmic approaches might be correct. If we know that a structure is rigid but movable, the possible correspondences are very different from the possible correspondences if the structure is not movable, but can be deformed. Using additional information for a particular application can help solve these problems. For a more in-depth treatment see [19, 72].



## 3. GPUs for High-Performance Computing

Since a large part of this thesis deals with GPU-accelerated algorithms (algorithms accelerated using graphics processing units), this chapter gives an overview over the evolution and the programming model of GPUs as high-performance coprocessors for scientific or medical computing, so that these concepts are in place for later chapters.

Algorithms in medical imaging problems often have challenging computational demands. On the one hand, the amount of available data is increasing, driven for example by a growing number of scans at higher and higher resolutions. On the other hand, there is a demand for intra-operative applications which require an interactive or near-interactive speed. Algorithms that are originally used offline, for example the registration of two scans of the same patient, are transitioning to interactive, possibly intra-operative usage. For offline computational jobs, a difference of even 100% in computation time might not have a large impact, since these tasks are performed as batch jobs anyway and inspected by the clinician once they are completed. However, the same difference in interactive use can have a large impact on usability. Faster computational speeds enable a more exploratory, interactive approach and allow the transition from off-line computation to interactive or even intra-operative applications. The increasing need for computational speed can either be met by algorithmic improvements, or by using special hardware in order to accelerate medical computing. GPUs have been established as a cost-effective means to incorporate parallel processing power for various applications.

### 3.1. Computing Coprocessors

Even before the wide availability of programmable, massively parallel coprocessors in the form of graphics cards, special purpose hardware has been used to accelerate demanding computational tasks. These acceleration options are still available and offer unique advantages but also disadvantages. Options for hardware-acceleration include custom boards using VLSI (very large scale integration), custom boards using FPGA (field-programmable gate array), or parallel computation on off-the-shelf hardware.

VLSI has been used for example for accelerating PET reconstruction [73]. However, custom-designed solutions do not profit as much from the scale advantage that more widely available hardware offers. A more flexible approach to hardware-based acceleration uses FPGA boards that have been used for registration [74, 75, 76], reconstruction [77, 78] and segmentation [79].

Compared to FPGAs, off-the-shelf parallel hardware offers a much easier programming model and all the advantages that come with the large market penetration [80]. In the following we will focus on this acceleration option.

When using off-the-shelf hardware, different parallelization options can be exploited, multi-processor parallelization, cluster-based parallelization or computing on graphics hardware. Note that cluster-based parallelization can be combined with computing on graphics hardware in clusters of GPUs as can be seen in the increasing showing of GPU-based super computers in the top 500 list as well [81]. Another recent trend is the blurring between computing on GPUs and multi-processor parallelization, with languages like OpenCL that support both options transparently.

Graphics hardware processing power has made huge leaps forward in the last decades and computing on graphics hardware has been of increasing interest for high-performance computing in the last years. Driven by the performance demands of the gaming industry, graphics cards have evolved from a fixed-function pipeline towards more and more programmability. Today, graphics processors can be considered highly parallel super computers on a chip. Today's GPUs outperform their CPU

counterparts by an order of magnitude, whether the computational power or the memory bandwidth are compared. This computational power can be used for many computational tasks that demand high performance, not only graphics rendering. Since a more direct access towards the computational power of GPUs has opened up, the research community has adopted new strategies for general-purpose computation on GPUs. For a better understanding of high-performance computing on GPUs, a brief overview of the evolution of graphics hardware and the demands that shaped its current programming model are presented, especially in contrast to the architecture of current CPUs. A discussion of the concurrent evolution of the programming languages used for GPUs follows, ending with a more in-depth discussion of the CUDA programming model that is applicable to other languages like OpenCL as well.

## 3.2. Rendering as a Parallel Problem

As the name graphics processor implies, the main function of GPU is the implementation of (or parts of) the graphics pipeline. The most common approach to real-time graphics is rasterization in which geometric shapes are converted into a raster image. Rasterizing in the graphics pipeline has been described as an "embarrassingly parallel" problem as it is inherently parallel and does not require very much effort to be separated into different tasks that can be processed in a parallel environment, compared to other computational tasks that require communication between parallelly executed tasks. The scenes that are rendered by the graphics pipeline consist of geometry data, the positions and other attributes of the vertices that build the geometric primitives, and the texture data. The texture data is image data that is pasted onto the geometric primitives to add detail to the appearance beyond geometric detail provided by the vertex information. The graphics pipeline processes this data in three main stages: the vertex stage, the geometry stage and the fragment stage.

The vertex stage transforms vertex attributes, for example the position or texture coordinates, then defines how texture data is applied to the geometry, etc. The geometry stage modifies geometric primitives like triangles, lines on the whole, or creates new geometric primitives. Finally, the fragment stage processes the attributes of a fragment like its color or its opacity.

All these stages have in common that they are amenable to parallel processing. The vertices in the vertex stage can be transformed without information about neighboring vertices, the geometric primitives can be processed independently from other geometric primitives, and the processing of a fragment does not need information about neighboring fragments. Therefore, all the primitives in the different stages can be processed independently of each other and in an arbitrary order. This programming model is called a data-parallel streaming architecture. Such an architecture processes all the elements of the input data independently but according to the same set of instructions. With the different stages in the graphics hardware becoming programmable the stage for the development from dumb graphics accelerators to a unified architecture of parallel computing cores was set.

## 3.3. Evolution of the GPU

In the 1980s, the first graphics accelerators [82, 83] were the domain of expensive super computers dedicated to graphics rendering. Much of the design of graphics cards today root back to the development at that time. In the late 1990s, the first consumer video graphics accelerators appeared, whereas before, graphics accelerators had been the domain of professional users who could afford the price tag. These graphics accelerators did not expose any programmability yet, but allowed to configure stages in a fixed rendering pipeline. Applications using the graphics accelerators had to be written using graphics APIs like DirectX or OpenGL [84].

In 1999, NVidia introduced the GeForce 256, the first graphics accelerator that implemented the complete OpenGL pipeline in hardware (and was DirectX 7 compliant). NVidia marketed this card as the "first GPU". The part that performed the graphics computations was named the Transform & Lighting engine (T&L). Another significant change to the previous generation was the introduction of



programmable register combiners that heralded the advent of the increasing programmability within the next years.

The evolution in the following years can be divided according to different releases of Microsoft's DirectX. In 2001, NVidia's GeForce 3 generation exposed for the first time the internal instruction set of the shader stages, supporting Microsoft's newly released DirectX 8. This release featured programmable vertex and pixel shaders, in an assembler like language. The first graphics cards to support this version were ATI Radeon 8000 and NVidia's GeForce 3 series [85]. In 2002, ATI Radeon 9700 introduced 24-bit floating point textures, then 32-bit floating point textures were introduced by the GeForce FX. This GPU generation implements shader model 2.x, removing limitations on the number of texture fetch indirections that had been in place before.

In 2004, DirectX 9 [86] was a major step forward with the Shader Model 3, removing shader length limits, and - particularly important for general purpose programming - introducing 32-bit floating point formats for textures. Examples of graphics cards supporting this version are the NVidia 6 and 7 series [87].

Additional features provided in this version are transform and feedback, texture fetches in the vertex shading stage (important for some GPGPU algorithms, see sections 5.3.1 and 5.3.5), and improved handling of branches and dynamic flow control in shaders (though to this day, avoiding divergent code branches in a shader/kernel remains an important optimization). This enabled the development of much more complex programs and reduced the necessity of multi-pass approaches.

In contrast to the assembler-like languages provided by the first shader extensions for OpenGL and the previous version of DirectX, high level shading languages were developed. Examples include the High-Level Shading Language (HLSL) by Microsoft and CG [88] by NVidia which were developed in cooperation but evolved differently in the following. HLSL is specific to DirectX, and therefore only available for Microsoft platforms, whereas CG supports both DirectX as well as OpenGL. The OpenGL equivalent of HLSL is the OpenGL Shading language (GLSL) [89].

The generation of the GeForce 6 and 7 still had separate hardware for separate stages in the graphics pipeline and reserved more computational power for the fragment stage as this was typically required in video games. An overview of the architecture can be seen in figure 3.1.

The next iteration of DirectX, DirectX 10 [91], improved on the preceding version in several aspects again. Geometry shaders were introduced that allow to manipulate geometric primitives, by transforming between different types of geometric primitives and adding or removing geometry in a shader, whereas the vertex shading stage can only transform the input vertices. Integer texture support and bitwise operations forced the hardware support of integer operations and automatic interpolation of floating point textures accelerated this operation compared to the programmatic solution in the shader.

Another difference was the unified shader interface. The different shading stages expose the same set of capabilities for example regarding texture access. While the graphics card architecture is not necessarily bound by the exposed interface, on the whole this approach leads to a unified architecture in the graphics hardware as well: the different stages in the graphics pipeline were now served by the same computing units. This was due to several factors: Load balancing the different stages was becoming more difficult, since there was no fixed load ratio between the different stages, so there was no good solution to this problem with a fixed relationship between the stages on the hardware either. The additional pipeline stage of the geometry shader only exacerbated the problem. Unifying the processing units enables a better utilization of the resources by dynamic partitioning for different load characteristics and avoided the problem that parts of the hardware were idle. The graphics pipeline visited the same processors up to three times for the different programmable stages.

The GPU in the XBox 360 introduced a unified design of the vertex and fragment stage in 2005. Another example of this unified architecture is NVidia's GeForce 8800. This architecture change allowed exposing the compute units as a general purpose computing resource in CUDA, since there was no distinction between the different pipeline stages anymore.

The current (at the time of writing) DirectX version 11 added additional pipeline stages for more control over the geometry detail by hardware-accelerated tessellation.

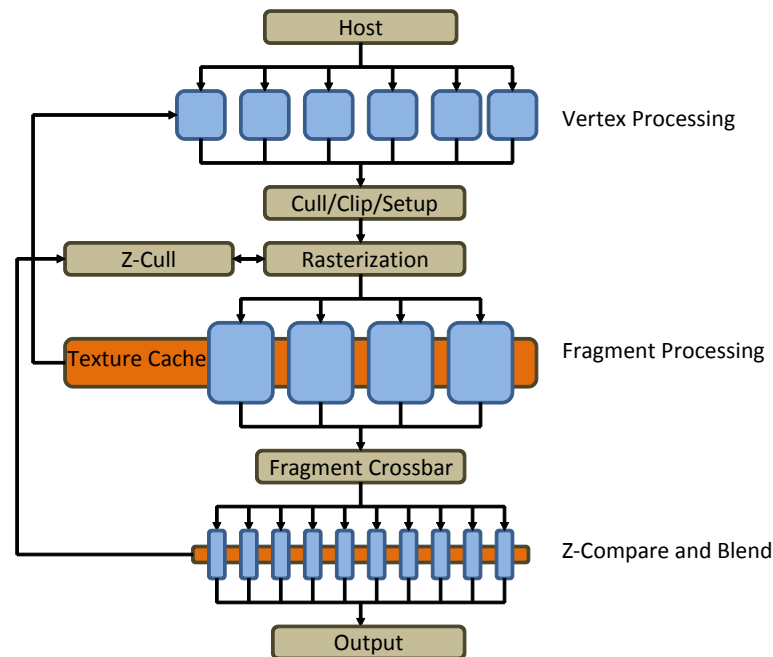


Figure 3.1.: The GeForce 6 architecture [90] uses different processors for different stages in the graphics pipeline. It also allocates more processing power to the fragment processing than to the vertex processing, since there are typically many fragments associated with only a few vertices. For this reason GPGPU applications use the fragment shader on this generation of graphics cards, with perhaps a few dependencies on the vertex shader for scattered write operations.

### 3.4. The Graphics Pipeline

In the following, we describe the graphics pipeline as it is implemented in today’s GPUs (see figure 3.3). For comparison, the pipeline for the first OpenGL version can be seen in figure 3.2.

The input to the graphics pipeline are geometry data in the form of vertices, their attributes and their connections (to form triangles, polygons or other shapes) and pixel data in the form of textures that are projected onto the geometric shapes for various effects. The following list explains the different stages of figure 3.3 in more detail. See also figure 3.4, for a visualization of the effects the pipeline has on incoming vertices.

- The vertex shader stage is the first stage in the pipeline and performs transformations of the per-vertex attributes, like vertex coordinates, texture coordinates and normals, usually from the world coordinate system to the camera coordinate system. It is not restricted to these basic transformations in modern GPUs anymore, though, but can perform arbitrary operations on the unified device architecture, even access texture data. Before the advent of more user-friendly interfaces to GPGPU programming, the vertex shader could also be used to perform scattered write (write operations to arbitrary positions in a target texture).
- Tessellation is an optional operation that consists of two programmable stages and one fixed function stage. The first programmable stage is the tessellation control shader (hull shader in

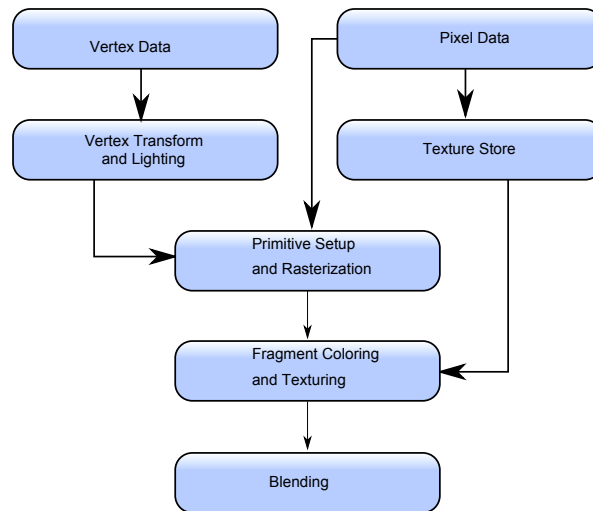


Figure 3.2.: This figure shows the graphics pipeline of OpenGL 1.0. All the stages have fixed-function functionality and do not expose any programmability to the developer. Instead the stages can be configured with different options.

DirectX). This shader computes per vertex and per patch attributes and is executed for each vertex, even though it operates on a patch-basis. The tessellation level can be adapted according to parameters like distance from the viewer etc. The fixed-function tessellation primitive generator then generates additional vertices based on the tessellation level specified in the previous shader. The tessellation evaluation shader is the third stage in the tessellation operation and perturbs the positions of the vertices programmatically.

- The geometry shader is another optional stage in the graphics pipeline. If a geometry shader exists, it operates on the primitive level and allows transforming primitive types from one to the other (for example generating triangles from points), allows inserting additional primitives or discarding unwanted primitives. Due to its limitation on the maximum number of stream out values (currently 1024 floats per instance) and the slower performance for higher stream out numbers, the tessellation shader has been introduced for the case of tessellation.
- The primitive setup and rasterization stage discretizes the incoming geometric primitives and generates fragments for each resulting pixel. Fragment attributes like depth, texture coordinates, color etc. are interpolated from the values at the vertices that build the incoming primitive. Early-Z test may discard fragments that fail the z-test at this stage already, however that is not possible, if the fragment shader introduces arbitrary changes to the depth of a fragment in the following stage. Usually the fragment shader computes the lighting on a per-fragment level taking into account the color, texture and position of the fragment in relation to the light source, but nowadays the fragment shader can perform arbitrary computations and output the result either to the frame buffer or to off-screen render targets.
- The programmable fragment shader stage determines the final color of each fragment by performing lighting calculations, texture mapping, procedural texture generation, etc.
- The fragment operations stage is a fixed function stage in the pipeline and performs the final blending of the incoming fragments with the fragments that are already in the frame buffer for transparency or antialiasing. Occluded fragments are discarded. This fixed-function stage creates

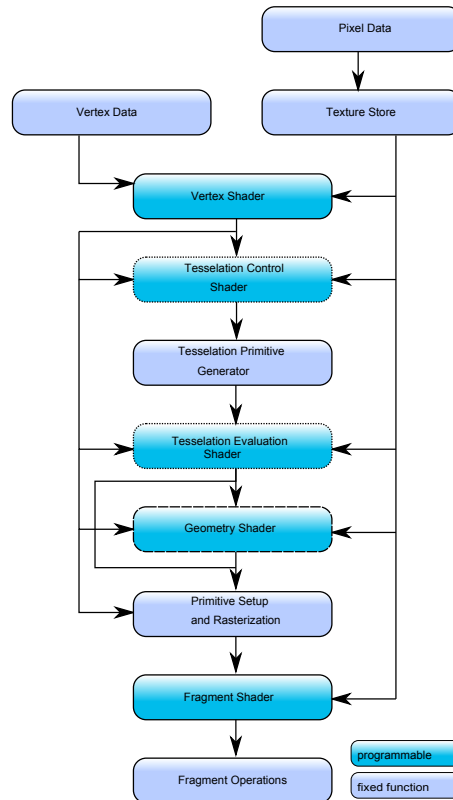


Figure 3.3.: This figure shows the graphics pipeline of OpenGL 4, as it is implemented on current GPUs.

the final color of the pixel on the screen, or in the texture if using FBOs. Other possible operations are stencil testing, scissor testing or logical operations.

### 3.5. Comparison of the GPU and CPU architecture

The graphics pipeline with its massive number of primitives like vertices and texels and its focus on parallel processing shapes the architecture of the GPU, compared to the architecture of the CPU that is geared towards sequential branching code, handling highly irregular data structures.

In order to accelerate sequential code, the CPU needs large caches that reduce the latencies in memory access, and sophisticated control logic that allows performing operations out of order for speed while still maintaining the sequential semantics. All these techniques require a large amount of die space, as can be seen in figure 3.5, but at the same time, they do not contribute to peak performance of optimized code. Parallelization by using several CPUs is often task-level parallelization in which different tasks are assigned to separate threads. This does not allow for an automatic performance scaling, since the number of tasks is fixed, and more computing resources require a reengineering of the software.

GPUs, on the other hand, are not suitable for executing sequential code, but are geared towards hugely parallel workloads. Compared to CPUs, the floating point logic and fixed function stages take most die space, whereas caches and control logic are very small in comparison with the CPU. In the GPU programming model, the larger latencies of memory access are hidden by switching to different threads. The GPU also has a programming model that is easier to parallelize, since there are no legacy

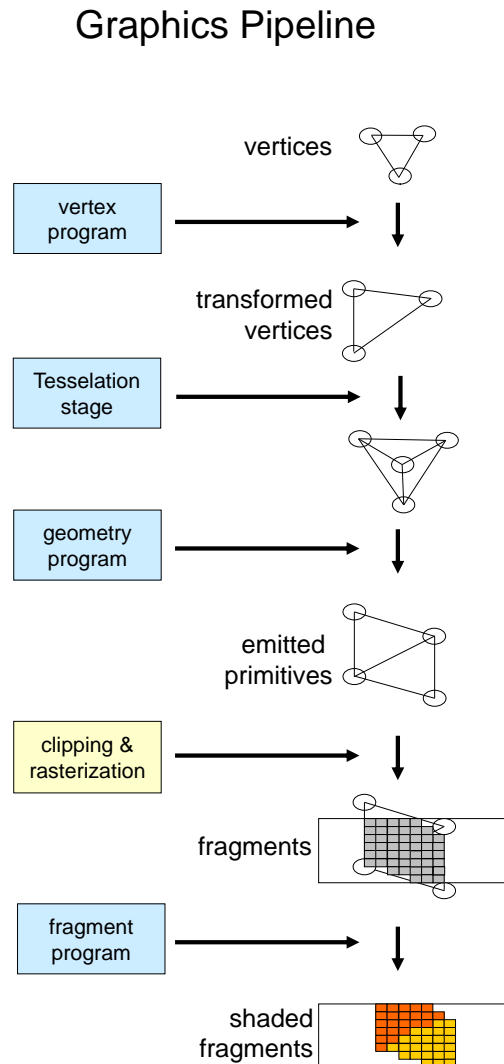


Figure 3.4.: This diagram illustrates how the incoming data is processed in the modern graphics pipeline.

constraints and no sequential semantics required of parallel programs. All this is due to the demands placed by modern games that require huge amounts of floating point operation for vertex transformation, attribute interpolation, etc. Therefore, GPUs are optimized for the execution of a massive number of threads. The fine-grained parallelism model of GPUs allows a more straight forward path to future performance gains just by adding more multiprocessors to the GPU.

These factors explain the gap in theoretical (and practical!) performance between the CPU and the GPU. Current CPUs and GPUs show this difference when considering the number of floating point operations per second as well as when considering the memory bandwidth. The NVidia GTX 680 for example has a theoretical GFLOP/s of more than 3000 and more than 190 Gb/s as theoretical memory bandwidth, compared to less than 500 GFLOP/s and less than 60 GB/s in the case of Sandy Bridge

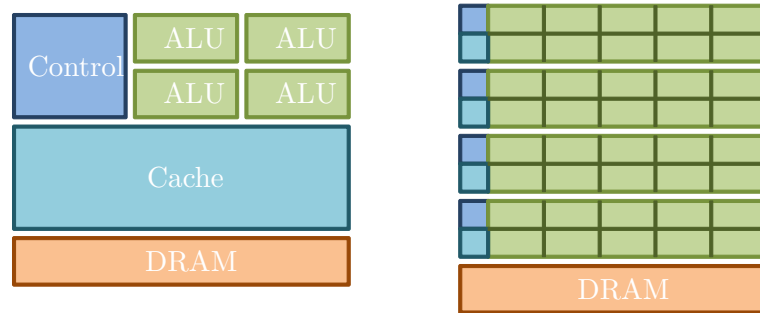


Figure 3.5.: GPUs and CPUs make very different trade-offs in order to accelerate code that is typically run on the respective device. While the CPU has complicated control logic for speculative and out-of-order execution of code as well as a large cache, the GPU devotes much more die space to the actual compute units at the cost of much simpler control logic and smaller caches.

from Intel.

## 3.6. High Performance Computing on the GPU

The computing power of graphics hardware due to the architecture decisions explained in the previous section has been realized early on and leveraged for general purpose computing. However, only at a certain level of programmability and speed, this area moved into the computing mainstream.

### 3.6.1. Using Graphics Libraries for General Purpose Computing

While the processors in GPUs are (also) marketed as number crunching machines today, they evolved from systems dedicated to rendering, and therefore the first programs using GPUs for scientific computations used graphics API, since GPUs did not allow a more direct programming. The fragment shader, as the unit that used to be the most powerful (before the introduction of a unified architecture) was the stage of choice.

Floating point textures are used as input as well as output memory. Using framebuffer objects allows directing the output of a rendering step to such an output texture instead of the frame buffer. The shaded fragments are the result of the rasterization process, therefore a quadrilateral that covers the complete output texture with a 1-to-1 mapping of fragments to pixels is used to invoke the fragment shader (see figure 3.6).

Several limitations in this model became apparent early: The output positions of the result of the computation are fixed, so only gather operations are supported but no scatter operations (at least not without additional effort). There is no communication possible between threads of operation on the different fragments, and the data has to fit into the supported texture formats.

In most cases OpenGL has been used as the underlying graphics both directly and for the implementation of the backend, though some authors have also used HLSL and DirectX [92] to the same effect.

### 3.6.2. Programming Languages for the GPU

Due to the inherent problems in using graphics APIs for general computations as well as the requirement for rendering knowledge, several languages were developed to make the development on a

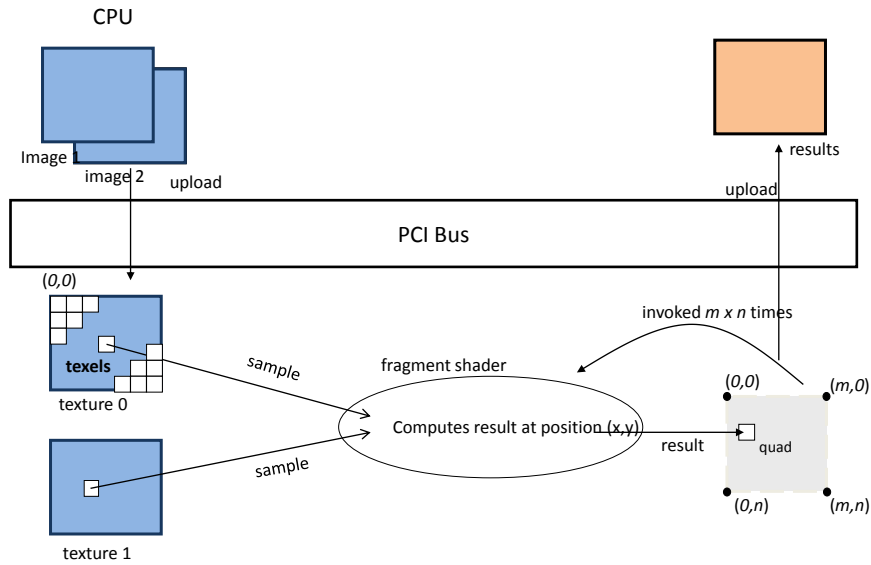


Figure 3.6.: This figure illustrates the concept of performing computations using a graphics API.

GPU easier for non-graphics experts. These languages, as well as the languages in use today, follow a stream programming model (see figure 3.7). In this model the input data is organized in streams consisting of discrete elements. The elements of the streams can be processed in arbitrary order (and therefore in parallel). The stream programming model is a good fit for the hardware architecture of modern GPUs (or processors like the CELL) [93], but it requires the programmer to define the parallelism explicitly.

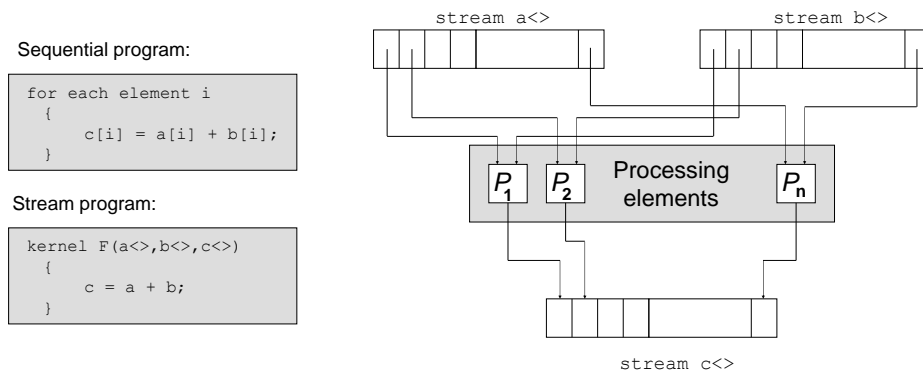


Figure 3.7.: This figure shows example code for a vector addition for the case of sequential code and parallel code on the left, and a diagram of the stream programming model on the right.

Initial endeavors to harness GPUs for general purpose computing with languages like Brook [94],

### 3. GPUs for High-Performance Computing

---

GLIFT [95] and SH [96, 97] are restricted since there was no direct exposure of the underlying computational resources, but all operations had to be implemented using a graphics API as a backend. Another alternative is skipping these abstractions and leveraging the graphics APIs directly. With the support of floating point textures in DirectX 9 generation hardware, this became possible for many problems in scientific computing. Examples of the diverse operations implemented in this way range from linear algebra computations [98], to sorting of data sets [99] or data base operations [100]. An overview of this research can be found for example in [101].

While this research proved the viability of general purpose programming on GPUs, due to the special knowledge of graphics APIs involved and the cumbersome interface, this approach remained a research exercise. However, the development of more appropriate interface to expose the computing power of GPUs for general purpose processing began.

The unified hardware architecture of the DirectX 10 generation graphics card allowed the exposure of the computing ability of GPUs via general compute APIs. AMD's CTM [102] is an early example using a very hardware specific assembly-like API, which developed into CAL (Compute Abstraction Layer [103]).

CUDA, introduced in 2006, was the first general purpose language that gained a more wide-spread acceptance for high performance computing on GPUs due to its more high-level approach to GPU programming and tight integration into existing developer tools. CUDA uses a SPMD (single program multiple data) model of parallel computing [104]. Another conceptually equivalent API is provided by OpenCL, initially developed by Apple and now maintained as a standard by the Khronos Group, which also maintains OpenGL. As in OpenGL, OpenCL consists of a core of capabilities and additional extensions that expose additional hardware-specific capabilities which can be integrated into the core functionality in the following versions. OpenCL not only supports GPUs, but is also available for CPUs. A third mostly equivalent language is Microsoft's Direct Compute for DirectX.

AMD - which had bought ATI - decided to abandon its previously developed proprietary interface in order to concentrate on using OpenCL [105] and DirectCompute instead. Currently, OpenCL, CUDA and DirectCompute are under active development (see also figure 3.8).

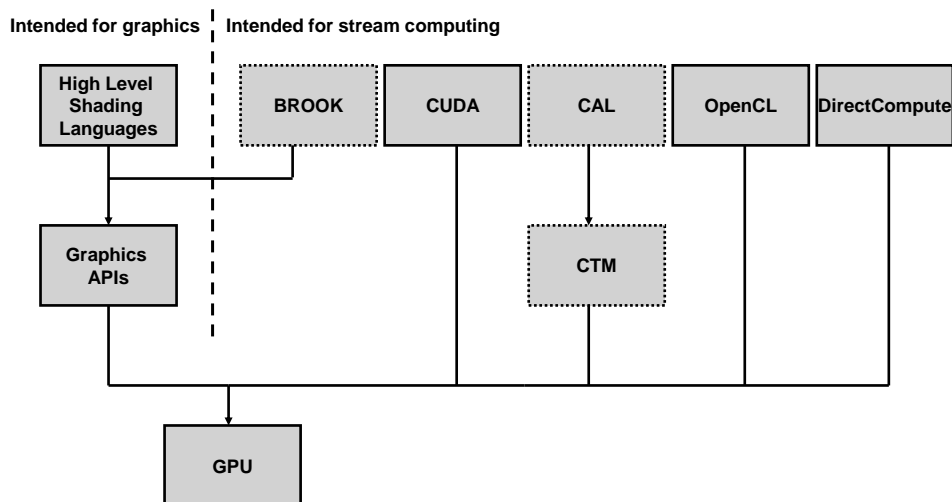


Figure 3.8.: Higher level shading languages communicate with the GPU using graphics APIs. GLSL (for OpenGL), HLSL (for DirectX) and Cg (for both OpenGL and DirectX) are examples for this. Languages intended for general purpose computing are on the right. Languages currently supported include CUDA, OpenCL and DirectCompute.



### 3.6.3. The CUDA Programming Model

The two APIs currently used for most GPGPU programming are CUDA [106] and OpenCL [107]. In the following CUDA is used for the overview, but the concepts apply equally to OpenCL, with only minor differences in terminology.

A thorough understanding of the GPU programming model is necessary for successful performance optimization on the GPU. While an understanding of lower-level implementation details is not needed for correct programs, it is necessary for performance tuning. This section provides an overview of the general programming model, as well as a discussion of implementation details based on the NVidia G280 hardware platform.

The figure 3.9 shows a conceptual overview of the CUDA programming model. This model is a direct fit for the architecture of the G80 graphics card (see figure 3.10). The unit of computation CUDA is a thread. These threads are very light weight compared to CPU threads and in order to fully utilize the GPU, thousands of threads have to be active at the same time. The threads are grouped in thread blocks, and the thread blocks in turn are grouped into a thread grid. Each thread has access to variables that hold its thread index and its block index, as well as the dimension of the blocks and the threads. These indices can be 3D in the case of thread indices and 2D in the case of block indices. The hierarchical organization of the CUDA threads is shown in figure 3.11.

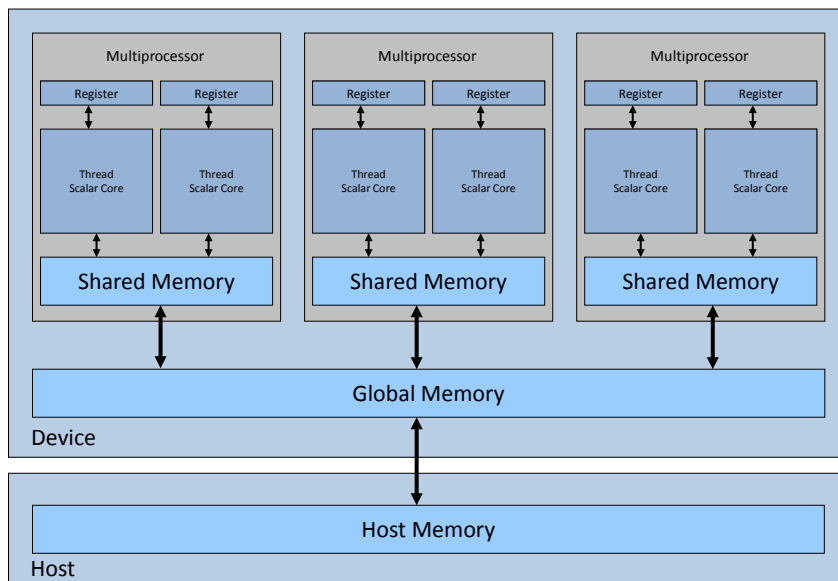


Figure 3.9.: This figure displays the components in the CUDA programming model. Streaming multiprocessors run blocks of threads. The streaming multiprocessors can be divided into scalar processors that run single threads. The shared memory is located close to the multiprocessor, so that fast access is possible. Registers, on the other hand, are private to the scalar processors. Host memory is the RAM accessible to the CPU, global memory is the RAM on the graphics card.

Threads of the same grid execute the same program, this is called the SPMD (Same Program Multiple

### 3. GPUs for High-Performance Computing

---

CUDA term	OpenCL term
thread	work-item
thread-block	work-group
shared memory	local memory
local memory	private memory
scalar processor	processing element
multiprocessor	compute-unit
GPU	device

Table 3.1.: This table gives an overview of CUDA terms and the corresponding OpenCL terms. Since the concepts are the same there is a one-to-one mapping from the CUDA terminology to the OpenCL terms. Unfortunately, the term "local memory" refers to different concepts in CUDA and OpenCL.

Data). Threads have access to different memory types:

1. global memory
2. constant memory
3. texture memory
4. shared memory
5. registers.

Global memory is memory on the GPU that is accessible to all threads and persists between running different kernels. Constant memory is accessible to all threads as well. An additional constant cache can accelerate access if the same constant is accessed multiple times. Texture memory provides automatic texture interpolation to threads. In contrast to these three memory types, shared memory is much closer to the processing cores on the GPU. Therefore access is an order of magnitude faster, but the amount of memory is very limited. All threads of the same block have access to the same shared memory, so shared memory can be used to communicate between threads of the same block, but not between threads of different blocks. Registers are closer to the processing units as well and the fastest type of memory, but only accessible by one thread.

Global memory has a much higher latency, but switching between threads serves to hide this latency. Whenever a thread has to wait for an access to global memory, it can be switched out for another thread that is able to perform work.

Since many threads are active at the same time, controlling competing access to memory is important. The default behavior for competing memory writes is that one of the writes succeeds, but it is undefined which one. Especially for read-modify-write operations this creates problems. Atomic functions provide a mechanism through which competing memory access patterns by several threads can be serialized. These functions read a memory location, modify it, and write the modified value back in an uninterruptible way. Competing atomic functions are performed in an undefined order. Beyond these general concepts, CUDA has several implementation concepts that help fully utilize the architecture. These concepts are unique to NVidia GPUs, but are applicable to OpenCL programs running on NVidia hardware as well. One such concept are so-called coalesced reads. This means that the full memory bandwidth can only be reached, if access to the global memory is in coalesced form so the currently active threads in a block access neighboring memory locations in a specific regular pattern. Another idiosyncrasy of the CUDA platform is the grouping of threads of a block into warps. These warps are consecutively numbered threads of a block that are executed at the same time by a streaming multiprocessor. For more details, the reader is referred to [109].

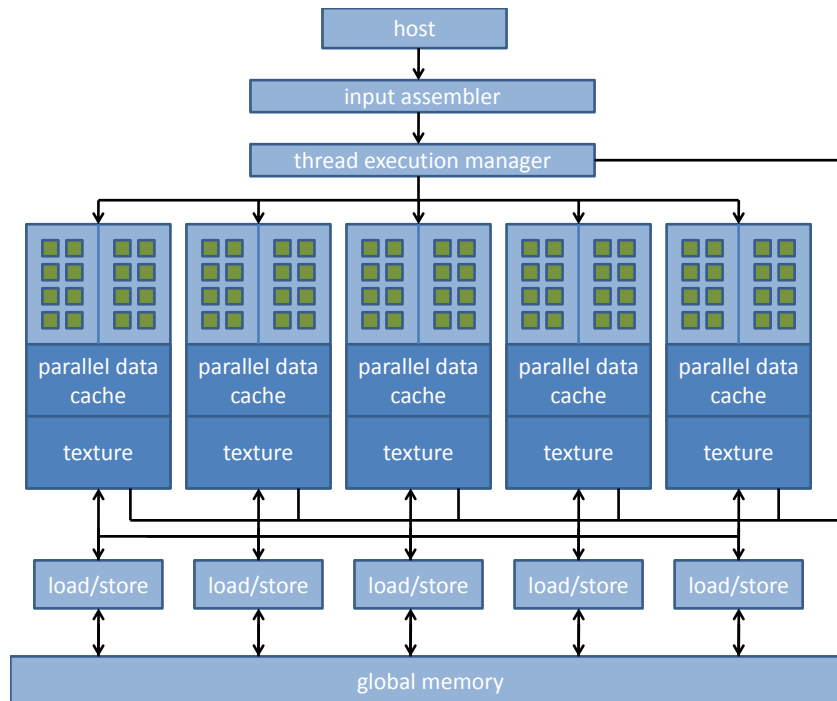


Figure 3.10.: This figure illustrates the architecture of a CUDA-enabled GPU (the G80) from a compute-centric view [108].

### 3.7. Acceleration Evaluation

With the increasing ease of use, especially of the CUDA programming model, a wide variety of scientific and medical applications have been ported to the GPU. While claims about extreme acceleration factors from porting applications to the GPU abound [110, 111, 112, 113, 114, 115, 92, 116, 117], these comparisons often suffer from several problems. Comparing an unoptimized version for the CPU to a GPU version that has been carefully tuned is not very meaningful. Another complicating factor is that the speed often depends on the exact work load. After careful optimization, GPU and CPU speed should get close to theoretical peak performance either for memory in the case of memory bound applications, or computing for compute bound applications [118].

### 3.8. Outlook

Parallel processing using GPUs continues to evolve towards more generality and programmer friendliness.

The Fermi generation [119] of NVidia's graphics offerings improve on previous generations in several aspects, from faster atomic operations, ECC (Error Correcting Codes) support for better error detection in high performance computing clusters, more shared memory to better double precision support. Several additional new features also improve the applicability of GPUs to a wider class of parallel algorithms. Faster context switches reduce the amount of data that is needed to be processed in order to use the GPU efficiently. The concurrent execution of kernels within one program is another feature that lowers the amount of processing necessary in order to benefit from parallel processing. A third interesting addition to this generation of cards is a true cache hierarchy that can be configured to

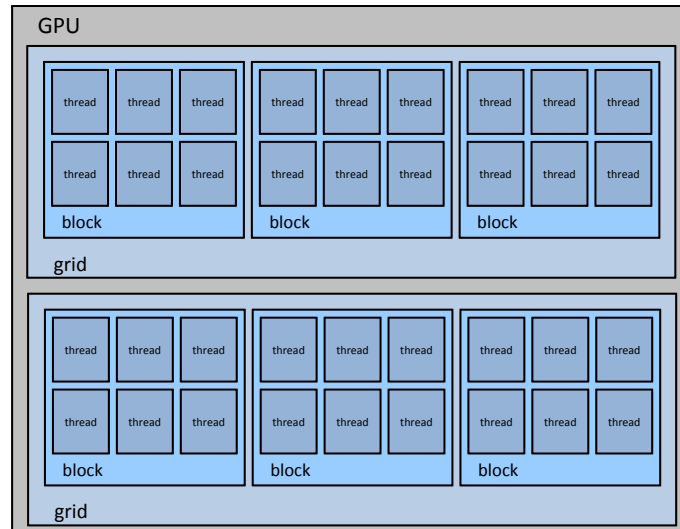


Figure 3.11.: Threads in CUDA (and OpenCL) are organized in a hierarchical fashion. Threads within a block can communicate via shared memory. Different blocks form a grid. Threads in different grids cannot communicate via that fine-grained messaging.

the application's needs. The effective use of shared memory enables impressive performance gain, but depending on the application the memory access pattern is not necessarily regular enough for using shared memory. The Fermi generation of graphics cards adds an L2 cache as well as L1 caches per streaming multiprocessor. The L1 caches and the shared memory on the streaming multiprocessor can be configured, resulting in either 16 Kb shared memory and 48 Kb L1 cache, or 48 Kb shared memory and 16 Kb L1 cache. Finally, the implementation of a unified address space generalizes the programming model and moves it closer to what is possible on the CPU. Previous generations of cards have three distinct address spaces, a global address space, shared memory per block and thread-private local memory. Because of this, the address space of a load or store operation has to be known at compile time. With a single address space, this is not necessary anymore, allowing a much more flexible use of pointers for example.

The following generation of cards is known as Kepler generation [120, 121] and introduces additional features that once again broaden its applicability to more algorithms. While there are other improvements as well, the most interesting addition in this generation is the dynamic parallelism. This feature allows to spawn new threads from within a kernel without an additional round trip to the host CPU, as was necessary previously. This allows producing additional workloads from within the kernel, especially for domains which have irregular computing patterns or need refinement in certain areas. Since there is no round trip involved, it also reduces the amount of parallelism needed for an efficient use of the GPU furthermore.

Additionally, coprocessors that are not derived from graphics cards, but from desktop processors are becoming available. Xeon Phi [122] is Intel's brand name for a multiprocessor computer architecture that is based on x86 processor cores. Same as with graphics cards, the coprocessors are hosted on an

extension card. In contrast to graphics cards however, the coprocessor can be programmed using the same tools that are used to program Intel's desktop processor. In contrast to the limited (though expanding) programming model provided by GPUs, Xeon Phi provides the complete programming model that is available on desktop processors (the coprocessor actually runs a linux variant). Programs that benefit from the parallel processing power of GPUs can be expected to benefit from the Xeon architecture as well. Additionally, optimizations for the Xeon like vectorization carry over to desktop CPUs.

### 3.9. Conclusion

GPUs keep evolving towards a more programmer-friendly and more general computing model. Programming languages for GPUs converge to the feature rich programming model of traditional CPUs, as can be seen in the support for the parallelization of different architectures, both CPU and GPU, in one language. OpenCL is the most widely supported option for programming not only graphics accelerators, but also exposing the parallelism of multiple cores for Intel and AMD processors. Other examples include C++ AMP by Microsoft [123] or OpenACC [124] which provides support for compiler directives that are modeled on those provided by OpenMP [125, 126]. GPUs are also becoming more common on mobile devices from smart phones to tablets. Efficient computing is becoming even more crucial for these battery-limited devices. On the whole, parallel processing in scientific and medical computing will stay an important topic, not only because of the demand for computational power and the good fit of many algorithms in medical imaging (for example registration and reconstruction as discussed in the following chapters) to a parallel approach, but also because improvements in sequential processing power are becoming slower and slower as has been pointed out earlier [127].



## 4. Review of GPU-Accelerated Registration

With the increasing computational power and more accessible programming presented in the previous chapter, medical image registration has also taken advantage of GPUs. In the following, we first discuss the implementation of the registration components from chapter 2, and then present a survey of technical papers about GPU-based registration. This chapter is partially based on [1]. Other recent articles on this subject are [128, 129].

### 4.1. Interpolation

Interpolation is the first building block necessary for medical image registration since it allows treating images as continuous functions. Fortunately, GPUs are particularly well-suited for image interpolation. Adding details to geometric primitives through the use of textures is one of the important steps in the graphics pipeline that improves the realism of the generated scene a lot. The interpolation of these textures when they are applied to geometric primitives is therefore hardware-accelerated and very fast. However, only nearest-neighbor interpolation and linear interpolation are supported natively [106], but more complex interpolation options can be implemented using OpenGL shaders or CUDA kernels. This can be necessary because linear interpolation might not be precise enough or because the performed low-precision interpolation introduces artifacts. Examples for custom interpolation schemes implemented on GPUs can be found in [130, 131].

### 4.2. Transformation

With interpolation in place, images can be transformed in order to find the transformation that best matches the two images. Rigid and affine transformations can be naturally mapped to the affine matrices that are used by the GPU for texture coordinate transforms. Projective and non-rigid transformations are more complicated and discussed in the following sections.

#### 4.2.1. Projection

However, using the GPU for registration involving projection is a natural fit, because the forward projection is the task of the GPU and there are highly tuned volume rendering methods. Radiographs that are used for generating a DRR [31] can be based on volume rendering algorithms [51], slice-based volume rendering [132, 133, 134, 135] or ray casting [136]. With the support of 3D textures, ray casting with trilinear interpolation for DRR generation becomes possible [137, 138]. Alternatively, Birkfellner et al. [139] use a method based on splatting to the same effect. A strategy for older hardware with lower frame buffer precision is presented in [140].

#### 4.2.2. Non-Rigid Transformation

The bulk of research on GPU-based registration has not been performed for affine or projective transformations, but for non-rigid registration algorithms due to its high computational demands. The following sections give an overview over relevant papers discussing non-rigid GPU-accelerated registration.

### Regularized Gradient Flow

To our knowledge, the first deformable GPU registration for 2D images was presented in 2003 by Strzodka et al. [141] and implements regularized gradient flow by minimizing the energy in the following equation.

$$E = \frac{1}{2} \int_{I_f, I_m} |i_m(\Phi(x)) - i_f(x)|^2, \quad (4.1)$$

where  $\phi$  evolves along the gradient

$$E' = (i_m(\Phi(x)) - i_f(x)) \nabla i_m(\Phi(x)). \quad (4.2)$$

This approach was extended to 3 dimensions by Kohn et al. [142], however the lack of hardware support for a render-to-3D-texture capability prevented this approach from reaching its full potential in speed. This facility was soon added in the DirectX 10 standard.

### Demons Algorithm

The Demons Algorithm [66] is an especially popular algorithm for GPU-accelerated deformable registration. The algorithm considers registration a diffusion process using an analogy to Maxwell's demons. These demons push according to local features in the image to be registered. These forces are created by the computation of optical flow [143]. The computation of optical flow is based on the assumption that the color (or intensity in the gray-scale case) of a point remains the same under small movements. Optical flow is used extensively in computer vision for example for motion.

The equation 4.3 is solved in order to compute the displacement vector.

$$\frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v + \frac{\partial I}{\partial t} = 0 \quad (4.3)$$

with  $I$  representing both the fixed as well as the moving image,  $u$  and  $v$  the unknown components of the displacement vector. In each iteration  $k$  the vector field  $u$  is updated for every voxel.

$$U(X)_k = U(X)_{k-1} + \frac{(I_m(\Phi(X)) - I_f(X)) \nabla I_f(X)}{(\nabla I_f(X))^2 + (I_m(\Phi(X)) - I_f(X))^2} \circ G_\sigma, \quad (4.4)$$

where  $U(X) = \Phi(X) - X$ . The resulting deformation field is regularized using Gaussian smoothing in each iteration.

In [144], the demons algorithm is implemented using BROOK, [145] uses the algorithm for registering MR images of the head. Muyan-Ozcelik et al. [146] and Samant et al. [147] use CUDA to implement the algorithm and compare their results to [145]. In [148], the authors implement five variants of the demons algorithm on the GPU and evaluate them for pulmonary 4D CT images using expert-determined landmarks.

### Hardware-Accelerated Free-Form Deformation

Another popular approach to deformable registration is based on free-form deformation. In 2001, Rezk Salama et al. [149] introduced volume deformation based on slice-based volume rendering accelerated by GPU support. The control points are vertices with associated texture coordinates, and the deformation is performed by changing the texture coordinates. The GPU then performs the linear interpolation automatically. This allows to keep the geometry static and does not require an intermediate volume representation. While dependent texture lookups for 3D textures were not available at the time, the authors already mention the use of this feature for volume deformation as it is commonly performed today. Soza et al. [150] compensate for brain shift effects for the registration of MR images using a piece-wise linear model for deformation. Extending [149] and [150], Soza et al. [151] use three-dimensional Bézier functions as basis function for a GPU-accelerated deformation. Fluck et



al. demonstrate in [152] GPU-accelerated free-form DRR for deformable 2D-2D registration for use in radiation therapy. The approach is based on accelerated GPU ray casting [136] and inverse ray deformation [153] and leads to a significant acceleration of the algorithm. In 2008, Li et al. [154] implement a non-rigid registration for Cardiac MR cine for point tracking that uses bi-cubic Bézier basis functions for image warping and for pre-calculation for finite element basis functions. [155] evaluates the accuracy of use GPU-based B-spline deformation for medical imaging applications and use it for elastic image registration [156]. The work in [157] present a parallelization-friendly implementation of the free-form deformation algorithms implemented and evaluated in CUDA.

### Physically-Based Registration

In contrast to the previously discussed deformation schemes which use heuristics for acceptable results, physically-based registration takes into account the physical properties of the imaged tissues in order to create realistic and physically plausible registrations, taking into account the different stiffness properties of different tissues, for example the difference between bone and soft tissue.

In [158], the authors employ a finite element model of the image that is registered. External forces that are computed by optical flow are applied to this model to arrive at a physically plausible deformation. However, the physical properties are not deducted automatically but have to be supplied by the user, via a fast, GPU-based segmentation process.

Noe et al. [159] present a registration method based on solving a viscous-fluid partial differential equation (PDE). They are solving the system of equations using a finite differences approach. Another reported approach uses optimal mass transport [160, 161].

## 4.3. Similarity Measures on the GPU

Similarity measures that compute how well two transformed images match are a good fit for GPUs due to the large number of features that have to be computed and compared. In general, the registration methods implemented on the GPU employ voxel intensity-based similarity measures, since the high computational demand and the regular memory access patterns are a good match for the GPU programming model. One exception to this is presented by Ruiz et al. [162]. The authors of this paper implement the registration of large sets of microscopic images with dimensions between  $16K \times 16K$  pixels up to  $23K \times 62K$  pixels. The registration has to compensate for the bending, shearing and other deformations that happen during the preparation phase. After a rigid registration for initialization using high-level features like blood vessels, features based on neighborhood complexity are selected for the following deformable registration. The cross-correlation between these features is the computationally most expensive part and therefore the part that is accelerated on the GPU. The cross correlation is implemented using Fast Fourier Transform (FFT) using the CUFFT library provided in the CUDA SDK [163].

All the other work on GPU-accelerated registration uses intensity-based similarity measures. Since these measures need to stream in every voxel of the fixed and the moving volume, they are very suitable for streaming architectures like the GPU. Examples of implemented similarity measures include sum of squared distances (SSD), sum of absolute distances (SAD), mutual information (MI) and its variants (MI), cross correlation and its variants (CC), and variances of differences (VOD). All these measures can be implemented with three basic building blocks: stream-wise operations, stream reduction and histogram generation.

SSD and similar similarity measures like SAD etc. are particularly well suited, since they compare values at corresponding spatial locations. The computation of the SSD does need interpolation though, since it requires samples that are not necessarily at grid positions. An additional reduction step is needed afterwards as well. SSD has been implemented on the GPU for example in [142, 154]. An example of a gradient correlation implementation on the GPU can be found in [138].

MI is another popular similarity measure for GPU implementation [164, 165],[2, 3]. The computation of the MI requires interpolation, stream processing, reduction and additionally the computation of joint

#### 4. GPU-Accelerated Registration

---

histograms on the GPU. The same is true, if the optimization method requires the use of gradients. The joint histogram can be built using nearest neighbor sampling by distributing the sample closest to the evaluated grid point into the histogram bin. With partial volume interpolation [37], the nearest samples are weighted by their interpolation weights and added to the respective bins of the histogram, as implemented on the GPU in [166]. Alternatively, the interpolated values can be added to the respective bins. The joint histogram can be considered a sampling of a probability density function. In order to estimate the true density function from these samples Parzen windowing can be performed as done in [167],[2, 3]. Parzen windowing is then performed by Gaussian smoothing on the joint histogram.

Table 4.1.: GPU Implementations of Common Similarity Measures

Measure	Reference
SSD	[142], [154], [59]
SAD	[59]
NCC	[162], [59]
MI	[164], [128], [2], [3], [165]

Kamen et al. [57] implement and compare different similarity measures on the GPU for the use in radiation therapy: LNC (local normalized correlation), GC, PI, GD, VWC, MI, CR (correlation ratio), and NCC. Another paper [29] implements and evaluates SSD, SAD, CC, RIU, PI, GC, GDGP (gradient proximity), SLNC, and VWC on the GPU. A further approach [168] computes the derivatives for the energy model based on different similarity measures by automatic differentiation applied to Cg code. The paper demonstrates the approach using normalized cross correlation. The authors in [59] implement 8 different similarity measures on the GPU and use the average of these measures for registration: SSD, SAD, VOD, RIU, NCC, GD, GC, and PI.

**Part III.**

**Contributions**



# 5. Learning-Based Multi-Modal Registration on the GPU

In this chapter, we present the GPU-based acceleration of a registration algorithm that uses additional information from previously registered images in order to perform a multi-modal registration. This chapter is based on two papers [2, 3]. In the following we will present a clinical use case this registration method addresses, explain the mathematical foundations of the algorithm and then detail the acceleration approaches in order to perform a successful port to GPUs.

## 5.1. Introduction: SPECT/CT Misregistration in Hybrid Scanners

Myocardial perfusion imaging is a diagnostic tool that allows assessing the function of the heart. It measures the perfusion of the myocardium, or how much blood reaches different parts of the heart muscle. It lets the physician diagnose cardiac problems and allows locating coronary stenosis. During the procedure, patients are injected with a radioactive tracer that is taken up by the heart muscle in proportion to its perfusion. The function of the heart can then be imaged under rest and under stress conditions using SPECT.

Since SPECT only shows the uptake of the radioactive tracer, but does not delineate any organs, a co-registered CT is useful in supplying this information. The CT provides the anatomical location of perfusion defects. Hybrid scanners like SPECT/CT (or PET/CT) scanners integrate both modalities in one scanner, allowing the acquisition of both low-resolution molecular images as well as the acquisition of higher resolution CT images at the same time. With this setup there is no transfer of the patient necessary and the two modalities are already co-registered since they are acquired in the same frame of reference. This setup does not only help with additional anatomical information, but also improves the SPECT imaging since the CT provides information for the attenuation correction of the SPECT (see chapter 7 for more details).

However, even when the two modalities are acquired at the same time, misregistration is still a problem, due to factors like patient movement, and exacerbated by the different time scales in which the two modalities are acquired. The resulting misregistration may lead to spurious perfusion defects and misdiagnosis. The American Society of Nuclear Cardiology (ASNC) therefore makes the quality control of the registration mandatory. For a more in depth analysis of the clinical significance of misregistration in hybrid scanners see [169].

An automatic registration between the CT and SPECT provides a potential solution to this problem by aligning the two scans correctly [170].

## 5.2. Method

In the following, we describe the components of a specific registration algorithm, designed to perform CT/SPECT registration.

The input to this algorithm consists of a CT scan and a SPECT scan, plus additional information on their alignment if available. Additional prior information is available as joint histograms of previous successful registrations.

### 5.2.1. Statistical Intensity Priors

Purely data-driven registration methods, like mutual information (MI) for example, have proven useful in practice and provide robust registration within modalities and especially between modalities. But even methods like MI suffer in practical clinical applications from a small capture range or misalignment in the case of data with high levels of noise or artifacts. Especially in the clinical case, artifacts are expected due to the generally older patient population that is more likely to have implants causing artifacts.

Incorporating additional knowledge into the context-free registration process in order to constrain the optimum of the registration to a clinically meaningful solution is a way to improve the registration result. In the following, we discuss information derived from joint histograms of correctly aligned registrations as a way to incorporate prior knowledge. Levonton et al. [171] introduce the use of joint histograms in such a way. Chung et al. [43] use Kullback-Leibler distance between the observed and expected joint histogram, resulting in an increased capture range compared to MI [172].

Guetter et al. [167, 173, 174] introduce using both MI as well as the Kullback-Leibler distance between histograms in order to perform multi-modal deformable registration. The joint histogram of previous successful registrations serves as statistical intensity prior that directs the optimization to an optimum that resembles these intensity priors. Using the MI criterion additionally ameliorates the problem that purely knowledge-based methods can only be as good as the provided priors and that inaccurate priors have a negative impact on the registration accuracy. Using both the divergence between observed and expected joint histogram as well as the criterion of mutual information is a compromise that improves capture range and accuracy even in the case of low-confidence priors.

Additionally, using a single prior is not sufficient in the case of medical imaging, since there is a large variability in the joint intensity histograms of correctly aligned images, depending on the exact anatomical location and the acquisition protocol, let alone different modality combinations. Instead a library of intensity priors can be used.

### 5.2.2. Cost Function

With this information we can develop a cost function that includes all the factors above. The cost function contains a term that computes the mutual information between the fixed and the moving image (the CT acts as the fixed image and the SPECT as the moving image in our case) given the current transformation, one term that computes the similarity between the joint histograms of the two input images and the learned histograms, and a regularization term. The regularization term  $\lambda$  determines the amount of regularization that is performed, whereas the term  $\alpha$  determines how much influence the mutual information has versus the statistical intensity priors.

$$\mathcal{J}(u) = \alpha \mathcal{I}_{data}(u) + (1 - \alpha) \mathcal{I}_{prior}(u) + \lambda R(u), \quad \alpha \in [0, 1], \lambda \in \mathbb{R}, \quad (5.1)$$

The data term  $\mathcal{I}_{data}(u)$  uses mutual information  $\mathcal{I}_{MI}(u)$  as defined in equation 2.13. In contrast to using a single joint histogram as in equation 5.2, the prior knowledge term  $\mathcal{I}_{prior}(u)$  uses Kullback-Leibler divergences to a set of histograms.

$$\mathcal{I}_{prior}(U) = \frac{1}{n} \sum_{j=1}^n \exp \left( -\frac{\mathcal{I}_{KL}(p_u, p_j)}{\sigma} \right), \quad (5.2)$$

with  $m$  the number of joint histograms provided.

The additional term  $R$  indicates the use of a regularizer. This prior imposes constraints on the deformation field. In this case, we use a simple smoothness prior, the standard Tikhonov regularization [175].

Following from equation 5.1, the minimization problem becomes  $\hat{u} = \operatorname{argmin} \mathcal{J}(u)$ . This minimization problem is solved by gradient descent using the derivative as computed in the following equation:

$$\nabla_u \mathcal{J}(u) = \alpha \nabla_u \mathcal{I}_{MI}(u) + (1 - \alpha) \nabla_u \mathcal{I}_{KL}(u) + \lambda \nabla_u R(u) \quad (5.3)$$

The derivative of the mutual information cost function is the following [176]:

$$\nabla_u I_{MI} = -\frac{1}{N} \left[ \left( \frac{\partial_2 p_u^o(i_1, i_2)}{p_u^o(i_1, i_2)} - \frac{\partial_2 p_u^o(i_2)}{p_u^o(i_2)} \right) * G_\sigma \right] (f_1(x), f_2(x + u(x)) \cdot \nabla f_2(x + u(x))) \quad (5.4)$$

\* is the convolution operator performed with the Gaussian  $G$ .

Similarly, the derivative of the Kullback-Leibler divergence is as follows [167]:

$$\nabla_u I_{KL} = -\frac{1}{N} \left[ \left( \frac{\partial_2 p_u^o(i_1, i_2)}{p_u^o(i_1, i_2)} - \frac{\partial_2 p^l(i_1, i_2)}{p^l(i_1, i_2)} \right) * G_\sigma \right] (f_1(x), f_2(x + u(x)) \cdot \nabla f_2(x + u(x))) \quad (5.5)$$

The terms necessary for the computation are the following:

- $p_u^o(i_1, i_2)$
- $p_u^o(i_2)$
- $p^l(i_1, i_2)$

$p_u^o(i_1, i_2)$  is the joint probability distribution of intensities in the fixed and moving input image.

This term can be approximated with the joint histogram and Parzen windowing [177]. The joint histogram contains point samples from the joint probability density function of the two volumes. This joint density can be estimated from the joint histogram using the Parzen-Window estimator.

$p^l(i_1, i_2)$  is the learned histogram that is provided as input.  $p_u^o(i_2)$  is the probability distribution of the moving and the fixed image and can be approximated same as the joint probability distribution. The second derivative of the distributions can be easily computed numerically.

The regularization term is implemented as Gaussian smoothing.

### 5.3. GPU Registration Pipeline

In the following we describe the implementation of the registration on the GPU as well as a hybrid implementation that runs on both CPU and GPU. We implement an iterative registration loop with the following basic steps.

1. computation of the metrics
2. computation of the gradient of the input image or volume
3. regularization of the gradient
4. computation of the displacement field
5. warping of the input volume according to the displacement field
6. computation of the metrics on the new warped image/volume.

These steps are run iteratively until either the metrics indicate that the quality of the registration is high enough or until the quality improvement of the last step falls below a predetermined threshold (see figure 5.1). In order to accelerate the convergence to the solution as well as to avoid local minima, the registration uses a multi-resolution approach; the solution of a lower resolution optimization problem is used as the starting point for the higher resolution registration process (5.2). As can be seen in figure 5.1, the GPU registration pipeline mirrors the CPU registration pipeline. The KL and MI metric are implemented according to Guetter et al. [173, 167]. Both use histograms on the GPU and Gaussian smoothing on the GPU, described in more detail in the section 5.3.1 and 5.3.2. A pixel shader computes the gradient on the volume. The gradient is smoothed as an approximation of the regularization,

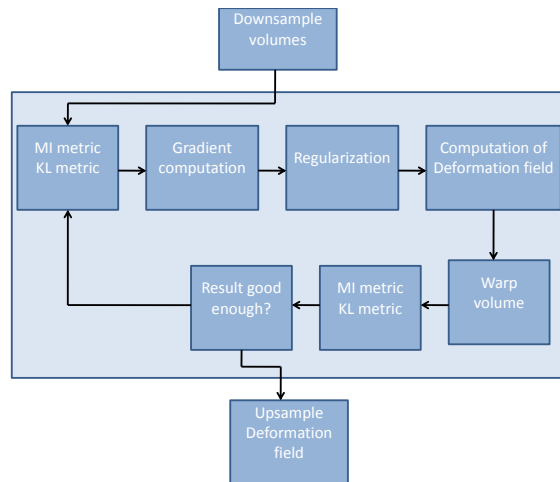


Figure 5.1.: This diagram shows the registration pipeline within one resolution level. The pipeline starts with the computation of the MI and KL metric. Then the gradient is computed and regularized, before the deformation field is computed. The moving volume is warped according to the deformation field, and the values of the two metrics are computed once again. If the result is good enough, the result is used as input for the computation on the next resolution step. The deformation field is up-sampled to the required resolution for this.

implemented as Gaussian blurring, described in more detail in the following sections as well. The regularized gradient is then used to compute a new displacement field, which in turn is used to compute a new warped volume by looking up the corresponding position in the input volume according to the displacement field and bilinear or trilinear interpolation. The entire registration pipeline shown in figure 5.1 can be implemented on the GPU with very little bus transfer between main memory and video memory, for operations on a small amount of data, however, it is often advantageous to download the data to the CPU. See also figure 5.3 for a visualization of the registration steps on actual data.

### 5.3.1. Histogram Computation on the GPU

As we can see in the following listing, computing the joint histogram is a trivial algorithm on a CPU.

**Algorithm 5.3.1:** GENERATEJOINTHISTOGRAM()

1. for (int i=0; i<number of elements;i++)
2. {
3.   movingValue=interpolate(movingImage,i);
4.   histogram[fixedImage[i]+bins\*movingValue]++;
5. }

On the GPU however, computing intensity distributions or histograms used to be an expensive step, due to the lack of efficient scatter as well as the lack of blending support for higher precision texture formats. The inherent problem is that addressing arbitrary memory positions and operating on the stored value destroys the independence between the processing of different elements. This is in



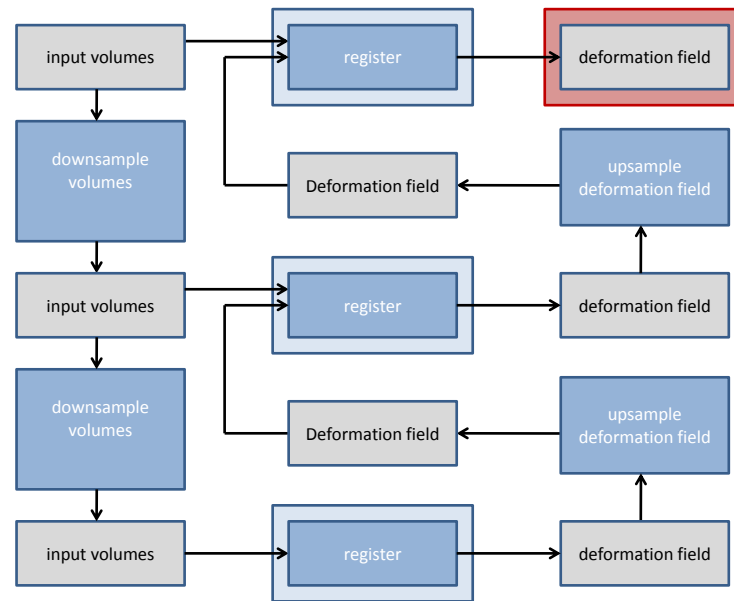


Figure 5.2.: This figure gives an overview of the registration pipeline on the GPU. Implemented is a multiresolution approach. The input volumes are downsampled for the lower resolution steps and then fed into the registration pipeline.

contradiction to the assumption on which the parallelization is based. The necessary synchronization makes the parallel processing more complicated and slower. One way to solve this problem exists for a small number of histogram bins, by casting the problem as a graphics problem. Occlusion queries are a method to determine whether an object is visible. The traditional approach to creating histograms on the GPU consists of using one occlusion query for every intensity bin and counting the pixels that are actually drawn. For two-dimensional histograms (histograms with the dimension 256 by 256 are used here), the cost for so many steps is prohibitive. So instead of occlusion queries, floating point blending and vertex texture fetches are used.

For every texel of the input texture a corresponding vertex is drawn. In the vertex program the output position of the vertex is shifted according to the intensities in the two input textures and rendered into a 256 by 256 render target. The color rendered is the floating point value 1. With blending enabled, the values are added in the render target.

The new generation of DirectX 10 capable graphics cards supports blending of 32-bit float textures as well as the use of integer textures, therefore the computation of the histogram can be performed in one step. On older graphics cards a more complicated approach has to be used, because only 16-bit floating values can be blended on hardware. The precision of this data type is sufficient for an addition by one only between the values of -2047 and 2048. Outside this range the numbers that can be represented lie farther apart than 1. To fully exploit the usable range, the render target is initialized with -2047. For each pass, less than 4096 values can be blended. Therefore a mesh with 64 by 63 vertices is shifted to access different parts of the input textures (see figure 5.4). The results of these partial histograms are then summed up into a 32-bit render target.

One-dimensional histograms are then computed by reducing the two-dimensional texture to a one-dimensional texture, while adding all the values during the reduction step.

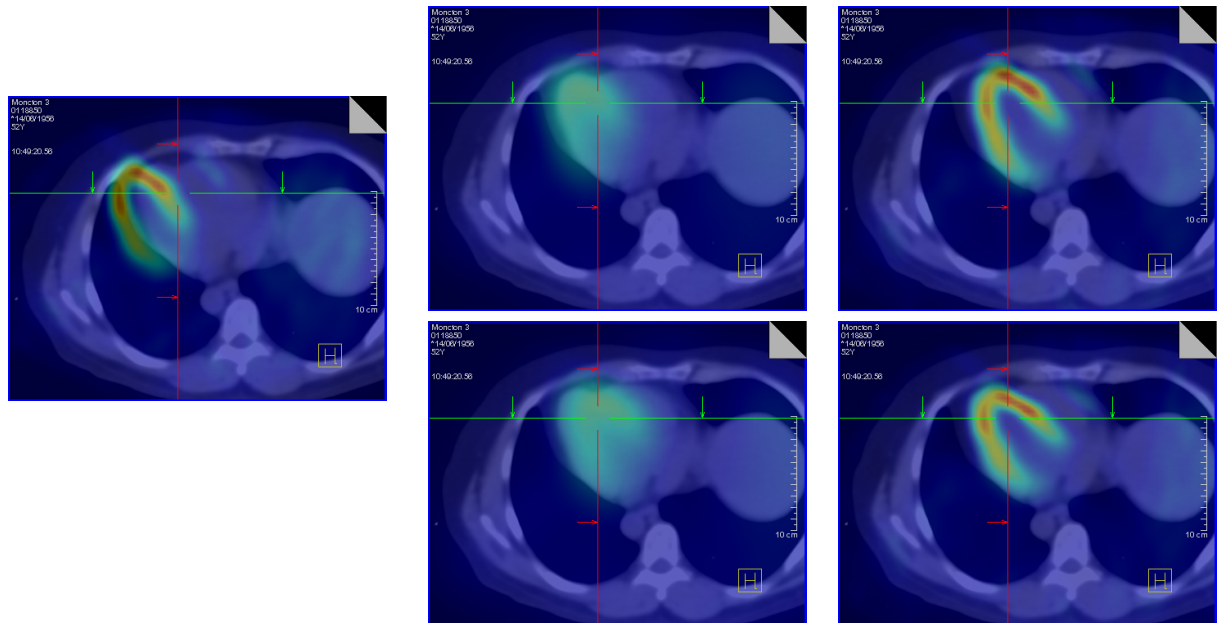


Figure 5.3.: This figure shows various stages in the multi-resolution registration pipeline. From top to bottom and from left to right, the images show the initial alignment of the SPECT and CT image, the alignment at the beginning of the coarse registration step, the alignment at the end of the coarse registration step, the alignment at the beginning of the fine registration step, and the alignment at the end of the fine registration step.

### 5.3.2. Regularization Using Gaussian Smoothing on the GPU

The next step in the GPU implementation that poses difficulties is the regularization step, implemented using Gaussian smoothing. For very small filter dimensions, the filter coefficients can be stored in uniform variables and the automatic interpolation of vertex attributes can be used instead of explicitly computing the position of the vertex sample. For somewhat larger but still relatively small filter dimensions, a brute force approach (that still exploits the separability of the Gaussian filter) is the fastest method. In an initialization step, a 1D texture with the filter coefficients is precomputed. A pixel shader convolves the input texture with the filter texture. For very large filter widths, this approach becomes inefficient. Gaussian smoothing on the CPU can be approximated by recursive filtering [178], thus making the computational cost independent of the filter dimensions. Furthermore, the data can be kept in the cache of the CPU, so that high speeds are possible. Due to the architectural differences between the GPU and the CPU, the smoothing has to be implemented differently on the GPU. Due to the dependency on the results of previous computation steps, the voxels cannot be processed independently anymore. Instead a line primitive is used. With this primitive, the input texture is scanned in x-, y-, and z-direction, and the smoothed result computed similar to the CPU implementation, but in parallel for all the pixels covered by the line. For smoothing in the x-direction for example, the line primitive advances from the left to the right, using the results that have been computed in the previous pass (that is by the previous line), in order to generate the values written by the current line to the texture.

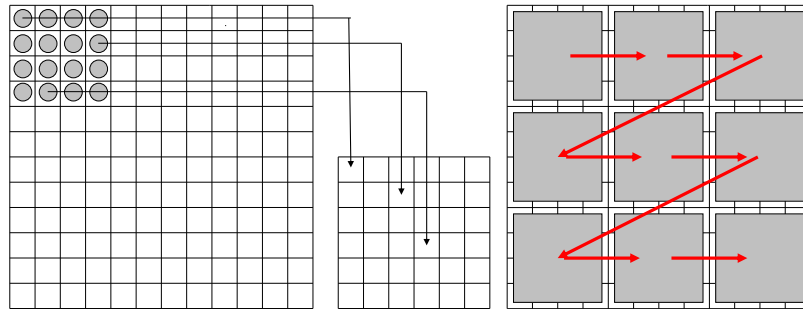


Figure 5.4.: Implementing histograms on the CPU is straightforward. The values can just be written into the corresponding bins. On the GPU, the implementation is less straightforward. The first possibility to implement histogram operations in OpenGL is using vertex texture fetches. The ability to access data in the vertex shader stage means that vertices can change their position depending on texture input. Using a regular grid of vertices, the texture is used as input, in order to perturb the vertices. With blending enabled, the result is written into a target texture as shown in the middle of this figure. If there is a restriction on the available bit-depth of the target texture in combination with blending, the operation can be performed in several steps and the intermediate result added as shown on the right.

### 5.3.3. Hybrid Implementation

With the histogram computation and the Gaussian smoothing in place, the remaining operations become straightforward fragment program implementations. However, depending on the size of the data, computations on the CPU can be preferable due to the overhead for the transfer between CPU and GPU. Two approaches for a hybrid implementation that uses both the CPU and the GPU have been developed. The first option is to run the lower resolution steps on the CPU, since there is not enough data to make use of the parallel computing power of the GPU, considering the overhead from shader and texture management. The second approach uses the result of the GPU implementation as starting point for the CPU implementation. This way, the GPU handles the many steps in the beginning and outputs a good estimate for the solution, which is then refined on the CPU using double precision for higher accuracy.

### 5.3.4. GPU Optimization

While the GPU implementation has a speed advantage over the CPU implementation already, further speed increases are possible when optimizing the data structures and algorithms to the GPU.

We employ the following optimizations:

1. store XYZ components of gradient in one texture instead of three
2. fixed loop number
3. 16-bit floats instead of 32-bit floats
4. histogram computation using VBOs.

In the first GPU implementation the three components of the displacement field - the x-, y-, and z-coordinates - are each stored in one texture. Instead of this, one can pack all three components into

one texture with three color channels. Since the same operation is performed on each component, this optimization increases the amount of parallelism available. The resulting speed increase however comes at the cost of higher memory requirements. Now, the texture holding temporary results during the blurring step needs three channels as well.

Conditional branches and dynamic loop can also have a negative impact on performance, especially in earlier GPUs that are not equipped to handle these efficiently. Using a fixed number of loops allows the shader compiler to optimize the instructions. Since a variable kernel width depending on the input is still preferable, the shader is recompiled whenever the filter width changes. Additional optimizations include exploiting the swizzle operator, improving vectorization, and pregenerating loop invariants as explained in more detail here [179].

Since memory bandwidth is an expensive commodity on the GPU, in many cases more so than raw computing power, we use 16-bit floating points for the textures instead of 32-bit floating points. This halves the required memory bandwidth, but does not result in a noticeable decrease in registration accuracy. These optimizations result in a speed increase by a factor of 3.7 for the smoothing step compared to the first implementation.

### 5.3.5. Optimized Histogram Computation

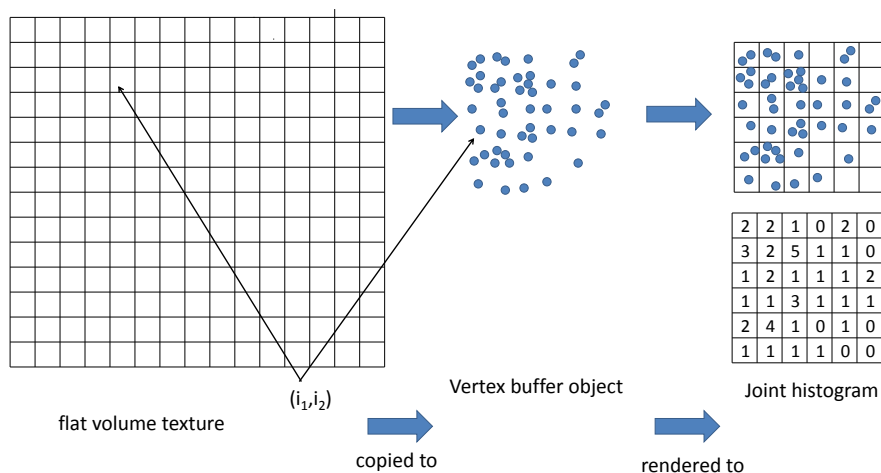


Figure 5.5.: The second approach to generating histograms in the OpenGL pipeline is using vertex buffer objects. Vertex buffer objects allow specifying vertex data in a buffer that resides in video memory instead of system memory and can be accessed very quickly.

The first implementation of GPU-based histogram computation has been implemented on GPUs of the DirectX 9 generation. The limitations of this generation of GPUs require a complicated multi-pass solution for the scattering and accumulation of the histogram values. The following generation of GPUs - supporting DirectX 10 - however provides VBOs (vertex buffer objects) and blending of 32-bit floating point textures. This allows the implementation of a more efficient solution using a single pass and therefore without the overhead of the previous multi-pass solution.

The intensity pairs of the input volumes are treated as coordinates of 2D points and written into the vertex buffer object. This cloud of points is then rendered into the joint histogram texture. The vertex processing stage of the rendering pipeline provides the support for the scatter operation. The associated color of each point is set to one so that blending accumulates the number of intensity pairs

written to each texel in the joint histogram texture. The absence of integer texture formats in this generation of GPUs limits the volume size that can be handled in one pass. While the floating point format covers a very large range of values, the histogram requires a representation of integer values. These integers however, can only be represented precisely between  $-2^{23}$  and  $2^{23}$ . Because of this restriction only volumes of size of up to  $2^{24}$  (for example  $256^3$ ) can be computed in one pass. With this restriction, however, this optimization results in a speed increase by a factor of 3 for volumes of size  $128^3$  and larger.

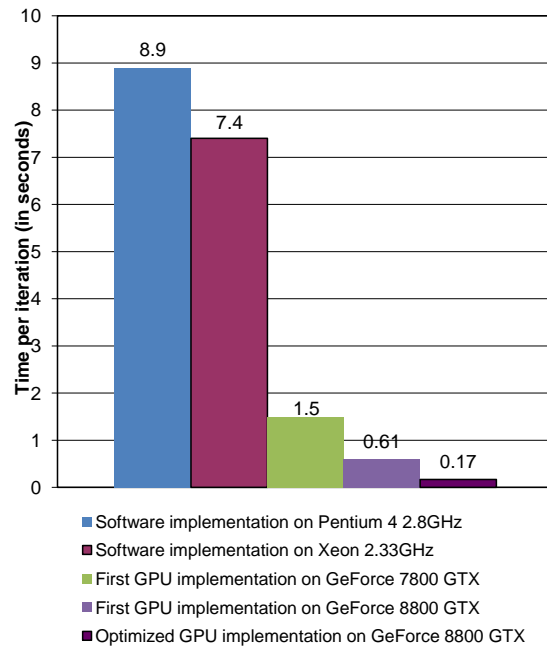


Figure 5.6.: This graph displays the various stages of the performance improvement for this project. The software implementation has been tested both on a Pentium 4 2.8 GHz as well as a Xeon 2 2.33 GHz. The original GPU implementation an NVidia 7800 GTX already improves on the performance of the CPU implementation. Further improvements specific to the 8800 further decrease the time for the registration even further.

## 5.4. Results

Figure 5.8 visualizes a registration result by showing the overlap of the CT and SPECT volume before and after registration.

### 5.4.1. GPU-CPU Comparison

Several experiments were performed using the GPU implementation. The first experiment uses a synthetic two-dimensional data set, a rectangle that is mapped to another rectangle. One iteration during the registration process is faster by a factor of 6 to 8 compared to the CPU implementation (at a resolution of  $512 \times 512$ ), yielding comparable results. In the next step, two multi-modal medical

## 5. Learning-Based Registration

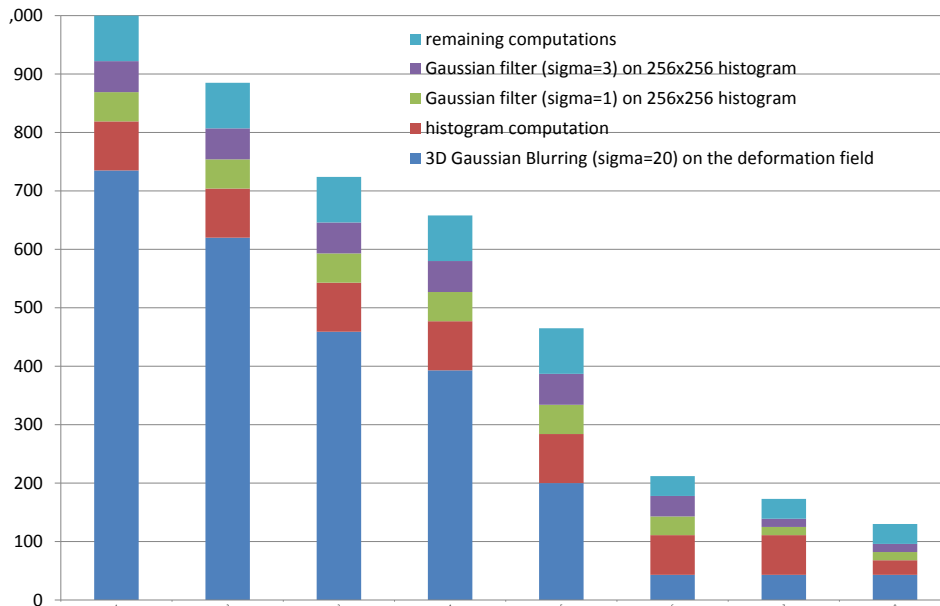


Figure 5.7.: This graph displays the effect of the optimization steps on the speed of the various components. 1) is the original implementation running on an NVidia GeForce 7800 GTX; 2) optimizing the 3D Gaussian blurring of the deformation field; 3) fixed loop numbers in the Gaussian filter; 4) using 16-bit floats for storing the deformation field; 5) storing the deformation field in 1 texture instead of 3; 6) upgrading to NVidia GeForce 8800 GTX; 7) accelerating 2D Gaussian blur; 8) changing the histogram computation to a single pass computation with a VBO and 32-bit floating point blending.

volumes are used, i.e. a CT volumes and its corresponding SPECT volume. These two datasets are registered using the CPU for the first two steps using one fourth respectively one half of the resolution and using the GPU for the final step at the full resolution. Iterations on the highest resolution take 22.95 s on average, whereas the GPU implementation needs about 3.9 s per iteration step. Surprisingly, the difference between the precision of the operations (floating point on the GPU, single precision on the CPU, or double precision on the CPU) results in only minor differences, whereas the quality of the learned joint histogram has a far bigger influence on the result, see [173, 167] for details. The measurements were performed on a Pentium 4 2.0 GHz with 1 GB of main memory, equipped with a NVidia 7800 GTX GPU.

### 5.4.2. Performance Optimization

The performance optimizations have been tested on an upgraded machine for the additional GPU capabilities, a Xeon 2.33 Ghz with a NVidia Geforce 8800 GTX. While the registration accuracy remains unchanged when optimizing the registration pipeline further for the GPU, the optimizations have a strong impact on performance. The performance improvements of the optimized version are tested on two input volumes of size 128x18x64 against the original GPU implementation. The results of our performance tests can be seen in figure 5.6. For a better comparison, the timings are measured in seconds per iteration. The original GPU implementation already improves on the CPU implementation from 8.9 s to 1.5 s. Just upgrading the GPU improves the speed to just 0.61 s and the final optimization

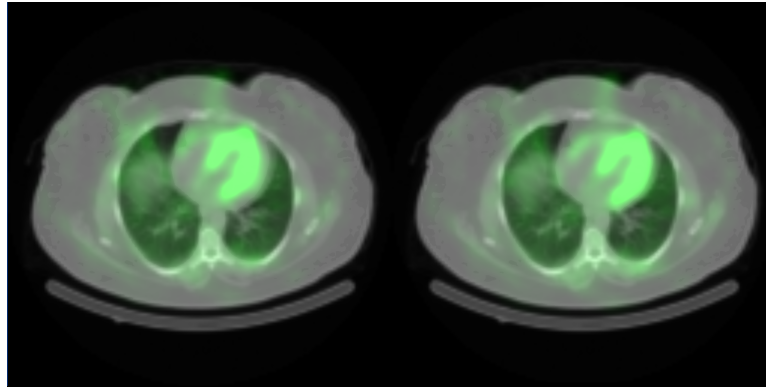


Figure 5.8.: These two images show the registration result: on the left side, the overlay shows a slice of the PET image on top of the CT image before the registration. Notice that the heart tissue in the CT and the activity in the PET do not align very well. The right image shows the same slices after registration, with a much improved alignment.

brings the time for one duration down to just 0.17 s, resulting in an increase of performance by a factor of 3.6 as result of the optimizations compared to the previous implementation on the same GPU.

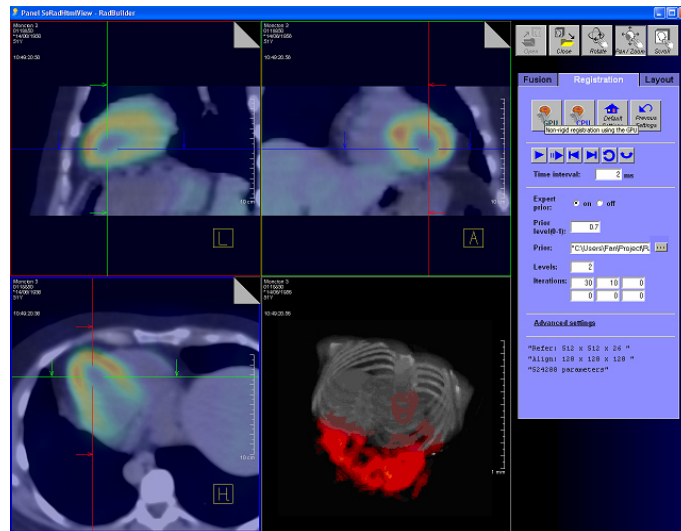


Figure 5.9.: This figure shows the screen shots of our prototype implementation with its integrated rendering and registration.

### 5.4.3. Integrated Rendering

Since the volumes used for registration already reside on the GPU, they can be easily visualized. In our prototype application in figure 5.9, we have integrated the registration algorithms that run on the CPU, on the GPU and the hybrid version. This allows a direct visualization not only of the

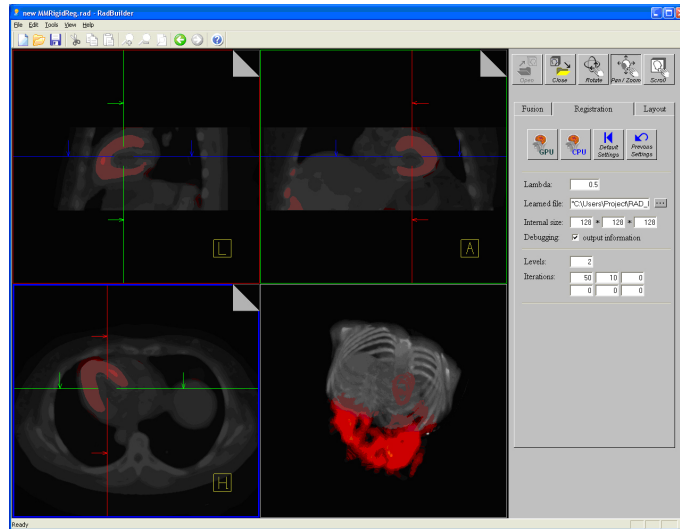


Figure 5.10.: This figure shows another screen shots of our prototype implementation in the development platform RadBuilder.

registration result, but also of every step in the registration pipeline. The user can interactively change the parameters of the registration and see the effects these parameters have on every step of the registration process.

### 5.5. Conclusion

This chapter has presented a highly optimized multi-modal registration algorithm that incorporates the information from previous successful registrations in order to guide the context-free similarity measure to a clinically correct solution. The integrated visualization also allows the user to interactively change the registration parameters as well as visualizing the registration process. While the optimizations performed in this case are purely using the capabilities of the underlying graphics library, the next chapter discusses optimizations using a general computing API for NVidia GPUs.



## 6. Optimizing Joint Histograms on the GPU

In this chapter, we present the careful optimization of the computation of the mutual information metric using the general compute API for NVidia GPUs, CUDA. This chapter is based on the paper [4]. As we have seen in the previous chapter, computing the mutual information between two volumes involves the computation of a joint histogram that is not well suited for the parallel architecture of the GPU and results in a complicated algorithm using the graphics API OpenGL. This still holds true for compute APIs, since the algorithm has a very irregular memory access pattern, but CUDA provides more optimization options than OpenGL. The main idea is to add a preprocessing step that is amortized over the following iterations. By sacrificing efficiency in some steps in the computation, a better memory access pattern can be used in later steps.

### 6.1. Introduction

#### 6.1.1. Computing the Joint Histogram

With the equation for the MI as given below (see also 2.13)

$$I_{MI}(U) = - \int_{I_f} \int_{I_m} p_U(i_f, i_m) \log \frac{p_U(i_f, i_m)}{p(i_f)p_U(i_m)} di_f di_m, \quad (6.1)$$

the term  $p_U(i_f, i_m)$  describes the joint probability distribution for the intensity of the two images. Given an intensity in the fixed image  $i_f$  and an intensity in the moving image  $i_m$ , the joint probability distribution gives the probability that these two intensities can be observed at corresponding positions in the two images. In order to compute the joint probability distribution, the joint histogram between the two volumes has to be computed first. The joint histogram counts how often each intensity combination between the fixed and the moving image occurs given a transformation between the two images. If the joint histogram is normalized by the number of elements in the overlap of the two images, it approximates the joint probability distribution. The following snippet pseudo code 6.1.1 shows a trivial implementation of a joint histogram computation as CPU code.

**Algorithm 6.1.1:** GENERATEJOINTHISTOGRAM()

1. for (int i=0; i<number of elements;i++)
2. {
3.   movingValue=interpolate(movingImage,i);
4.   histogram[fixedImage[i]+bins\*movingValue]++;
5. }

#### 6.1.2. Computing Histograms on the GPU

A naive implementation on a GPU is not much more complicated than listing 6.1.1 above, but such an implementation suffers from a low performance. Implementing this histogram procedure efficiently poses a challenge, especially on the GPU. The histogram is characterized by write access to arbitrary memory locations depending on the content of the two images that are used to generate the joint histogram. This is extremely inefficient when implemented on streaming processors directly.

Before the advent of programmable stages in the graphics pipeline, a special OpenGL extension [180] supported accelerated histogram generation and has been used for example in [181]. This extension is not widely supported anymore, though, and the more generally applicable programmability has superseded it.

In the graphics pipeline, the use of vertices enables scattered write access. While in the GPGPU paradigm on graphics libraries the fragment stage was preferred due to its higher throughput and easy mapping, the fragment stage is not built for scattered write operations. Using vertex texture fetches, that is fetching data from a texture in the vertex stage in order to offset vertices, allows scattered writes for histogram computations in [2] (see also chapter 5). Another option is using vertex buffer objects as in [3],[182] (This approach has already been explained in more detail in chapter 5.). This technique allows the interpretation of texture data as vertex positions.

Scattered writes to arbitrary positions are not the only complication though. The other problem are colliding write access operations to the same memory address. In the graphics pipeline this is actually a common problem, for example when using non-opaque primitives. This is then solved by blending the incoming fragment with the fragment in the buffer and results in well-defined behavior.

Another approach demonstrated in [183] converts the scattered writes into a gathering process at the cost of greatly increased number of texel fetches. Scheuerman et al. [184] compare different histogram algorithms using the graphics pipeline.

### 6.1.3. Compute APIs for Histograms

Compared to graphics APIs like OpenGL and DirectX, compute APIs for GPUs open up new possibilities. In CUDA and OpenCL write-operations to arbitrary memory addresses are supported directly, but conflicting write operations are serialized by discarding every operation except one. Which operation succeeds exactly is undefined. Atomic write operations solve this problem. These operations perform atomic read-write operations that are guaranteed to succeed, but the exact order is once again undefined, and the necessary serialization implies a performance cost. Before the support of atomic operation in shared memory, these operations had to be simulated [185].

Since streaming processors have huge speed advantages for many image processing tasks in the medical domain, and computing histograms is an essential part not only for the registration using the mutual information or Kullback-Leibler, but also many other algorithms, there have been a variety of approaches to implement the histogram in a GPU-friendly way. The naive method is to just stream in both images from global memory and perform atomic operations on the global memory to increment the counters in the histogram. Only considering the read access, this approach is a good fit for the architecture of GPUs, since elements are read in a GPU-friendly manner and every element is read exactly once. However, write access to the same memory location has to be serialized, and whenever a counter is incremented, there is a round trip from global memory to the processor core and back to global memory. Additionally, the write pattern is very irregular so that the operations are not coalesced. This makes the operation slower than necessary. Due to the speed of streaming hardware for imaging tasks in other parts of the pipeline, the overall speedup still results in a net acceleration, but the histogram computation becomes a bottle neck. Depending on the exact number of bins in the needed histograms, threads can either allocate a sub-histogram or have to share a sub-histogram with other threads in the warp [185, 186].

Podlozhnyuk [185] describes an approach which uses subhistograms in the much faster shared memory. These subhistograms are then merged into one histogram in global memory. The subhistograms are either allocated per thread and the number of bins is limited to 64 on G80 hardware due to the size of the available shared memory, or the subhistograms are allocated per warp, then the histogram can contain up to 256 bins, but at the cost of inter-warp collisions when writing to shared memory.

These optimizations do not solve the problem for joint histograms, though. The number of bins for each image is often larger than 100, and 256 bins per image is a common use case. 256<sup>2</sup> bins are too much for the limited amount of shared memory though. Shams et al. [164, 186] solve this problem by dividing the histogram into several subregions that are computed in separate passes. This allows the

algorithm to use fast shared memory, but it has the disadvantage that the two volumes have to be streamed in from the slower global memory several times.

Chen et al. [187] perform a preprocessing step that sorts one of the input images. After that step, specific regions of the sorted image contain only specific values. This reduces the number of possible values in the subregions, so that a temporary histogram fits into shared memory again. The histograms of different regions are then combined into the final histogram. This approach suffers from shortcomings in the hardware it was presented on, since the hardware did not yet support atomic operation in shared memory, therefore these operations had to be simulated. A closer analysis later in this chapter will also reveal how competing atomic operations degrade performance severely, when they have to be serialized. But even if atomic operations are available, resolving conflicting write operations still costs time.

Brosch et al. [188] present another example of an algorithm that performs a preprocessing step. Their aim is to reduce the number of conflicting write operations by precomputing an optimal number of memory locations for each histogram bin in shared and global memory. This allows reducing the number of write conflicts by spreading out counter updates for a specific bin over several memory locations. But the duration of this preprocessing step on the order of several seconds prevents its use for interactive registration.

In 2010, Shams et al. introduce the sort-and-count algorithm [189]. This algorithm performs a stable customized sorting operation. Every element has an associated counter. When two equal elements are compared, the counter of the element with the higher index is incremented by the counter of the element with the lower index. The counter associated with the element of the lower index is set to 0. Since the algorithm is stable, there are no inversions in the order of equal elements. After the sorting the last element in a sequence of equal elements contains the number of these elements. These values can then be gathered into the final histogram in global memory. This approach avoids the colliding access pattern of previous approaches, but the sorting step is costly. The complexity of the sorting is on the order of  $O(n \log n)$  higher than the complexity of  $O(n)$  for the histogram algorithm. Despite that the algorithm is still the fastest algorithm in practice for large bin sizes.

## 6.2. Method

Even considering all the previous research, computing joint histograms can still be improved by a careful analysis of the bottle necks and the trade-offs involved. Generating histograms on the GPU suffers from three major bottlenecks:

1. The random write access to memory locations is not a good fit for streaming processors that rely on coherent access to memory regions for speed.
2. Furthermore, updating the counters at random positions involves a read-modify-write operation when several threads compete to update the same counter, since access to the same memory address has to be serialized for a correct result. Especially when this is done in global memory, the loss of speed is considerable.
3. The restricted amount of shared memory makes it impossible to keep the whole joint histogram in shared memory.

An improved algorithm is based on two ideas: We use a sorting step similar to Chen [187] and reduce the number of conflicting atomic operations using a higher number of memory locations per bin. The preprocessing step adds a one-time cost at the beginning of the algorithm, this cost however is amortized during the following evaluations. Clearly, this approach cannot be used if the joint histogram is used only a few times, but in the case of registration, the metric is typically evaluated hundreds of times. Using the preprocessing step is a trade-off between two conflicting goals, coalesced read access and coalesced write access. The sorting destroys the spatial coherence of one of the input volumes, but it allows us to write the resulting histogram in a coalesced way and helps avoid conflicting atomic operations on global memory completely. Furthermore, the number of possible values for the joint histogram in specific regions is significantly reduced, so that partial histograms fit into shared memory

completely and the two volumes need to be streamed in from memory only once. The cost of this preprocessing step amortizes itself very quickly if the joint histogram has to be computed many times as is common in registration. When the joint histogram between two images is computed only once, this approach is clearly not applicable.

The second idea is based on a probabilistic distribution of the number of bins per intensity of the fixed volume. This allows us to reduce the number of conflicting operations on the shared memory.

### 6.2.1. Preprocessing

In the preprocessing step, the fixed image is sorted according to its intensities. This guarantees that in a continuous region of the moving image, there is only one intensity left. Combined with  $n$  intensities in the fixed image (which is not changed), the histogram for that region needs only  $n$  bins instead of  $n^2$  bins.

The position of each voxel in the fixed volume is recorded in a second volume, then the position volume is sorted with the associated intensities in the moving volume as key. This results in so-called runs of identical intensity values in the sorted moving volume with the original position of the intensity value stored in a position volume. The runs in the new volume are aligned to boundaries of  $k$  times the number of threads in a block. Each aligned part of a run is in the following called a line. Since the number of intensities of one specific value is not necessarily aligned to the number of threads in a block, lines that are too short are filled with a specific invalid value marker. These lines are in the following processed by separate thread blocks. The pseudo code in the listing 6.2.1 shows an overview of the preprocessing step and figure 6.1 shows a small example.

**Algorithm 6.2.1:** PREPROCESS(*fixedImage*, *movingImage*)

1. for (int i=0; i<number of elements;i++)
2. {
3.   fixedPositionData[i]=i;
5. }
6. sort fixedPositionData, fixedImage intensities  
   with fixedImage intensities as key
7. write out positions of runs in (sorted) fixedImage
8. align fixedPositionData to thread block boundaries

The sorting step can be implemented efficiently on the GPU using optimized sorting routines as provided by the CUDPP library [190]. On the sorted image, the starting positions of runs of equal intensity are determined as well as the length of each run. With this information, offsets are computed that align the runs to boundaries of thread block size. In practice, only the position image is aligned to these boundaries. The fixed image is not needed anymore and can now be discarded, since only the intensities for each line after the alignment operation are needed anymore.

### 6.2.2. The Histogram Kernel

After this preprocessing step, the position data for each line points to the same intensity values in the fixed image, so if each line is processed, only the intensities in the moving image vary. Reading the position data is fully coalesced for optimal performance. Each line can now be processed independently and transformed into its histogram using an algorithm that uses shared memory to hold partial histograms (see figure 6.2). Competing atomic operations on the same memory address cannot be avoided at this time, but we will show in the following section how they can be reduced. Competing atomic operations on global memory are completely avoided, though. We further reduce the number of writing operations to global memory, by assigning each thread block consecutive lines to work on (see figure 6.3). As long as the intensity value of the following line is not different from that of the current

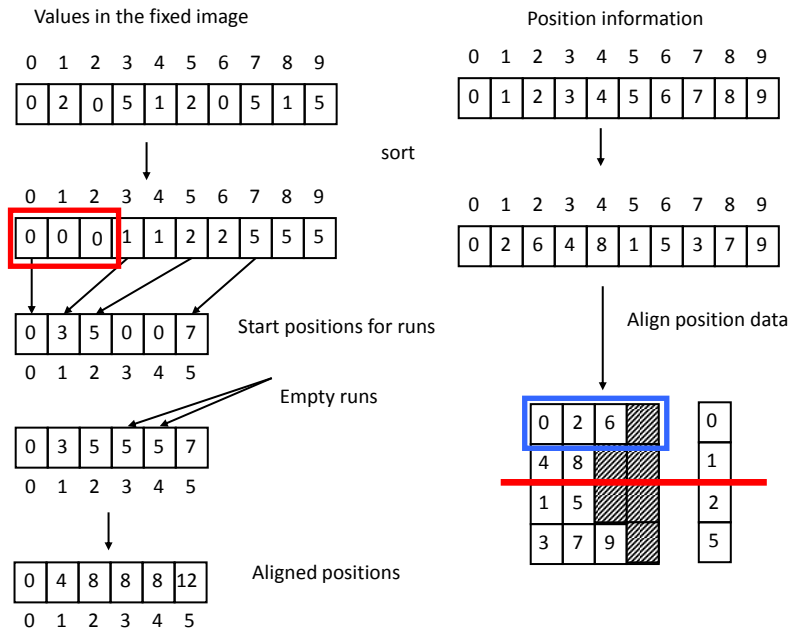


Figure 6.1.: This figure shows a small example for the preprocessing step. On the left side are the intensities in the fixed image with their associated position information, a linear index into the image. The right side shows the image containing these indices. After the sorting step, the fixed image contains runs of identical intensity values. One example run is marked by the red box. The position image holds the original positions of the associated intensities in the sorted image. The start positions of each run are now determined, empty runs have 0 as their start position. The positions of empty runs are then corrected to the start position of the preceding non-empty run. In a final step, the start positions of the runs are aligned to the chosen thread block size. In this example, we have chosen a thread block size of 4 for illustrative purposes, but in the implementation the size is 256. With this information, the position data is aligned to these boundaries. The blue block shows the position data of the run in the red box after alignment. Invalid value markers are displayed as hatched rectangles in the diagram.

line, the partial histogram can be kept in shared memory, otherwise the partial histogram needs to be written to global memory and a new partial histogram initialized to 0 in the shared memory. A simplified pseudo code for this step that does not show writing to global memory is shown in listing

## 6.2.2.

**Algorithm 6.2.2:** JOINTHISTOGRAMCUDA()

1. for all threads in parallel
2. compute current position
3. `offsetPosition=positionLookup[position];`
- 4.
5. clear shared memory
6. synchronize threads
- 7.
8. `data = movingImage[offsetPosition];`
- 9.
10. `sharedMemOffset=offset[data]`
11. `threadIdx.x%counters[data];`
12. `atomicAdd(s_Hist + sharedMemOffset, 1);`

With these transformations, there are no uncoalesced writes to global memory anymore. The read access, however, follows a more random pattern now, because of the reordering according to intensity values. The reading still maintains some locality since intensities of the same value are often close to each other in natural images. Using a stable sorting algorithm preserves some of that locality.

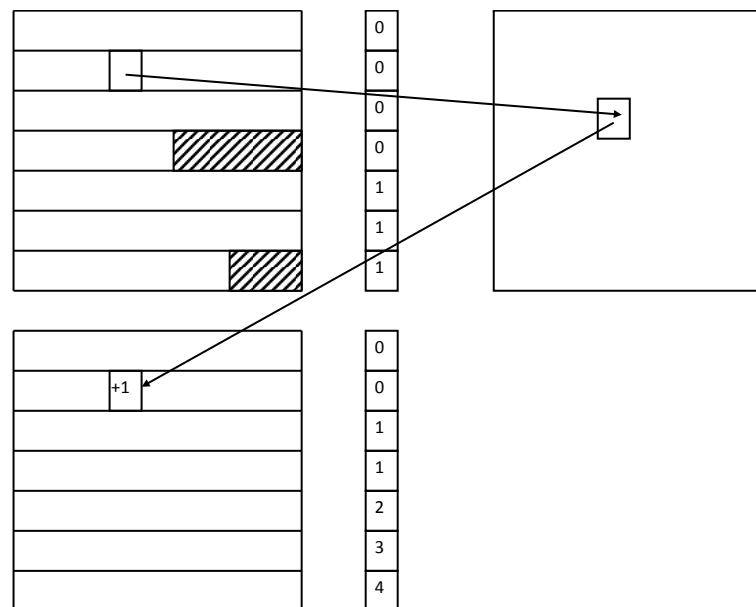


Figure 6.2.: The computation of the histogram is now performed by processing each line and generating the histogram for this line. The intensity value of the fixed image is the same for every element in the line, only the intensity in the corresponding positions in the moving image can vary. For each element in the line the corresponding intensity in the moving image is looked-up using the position data. The counter corresponding to the intensity value of the moving image is then incremented.

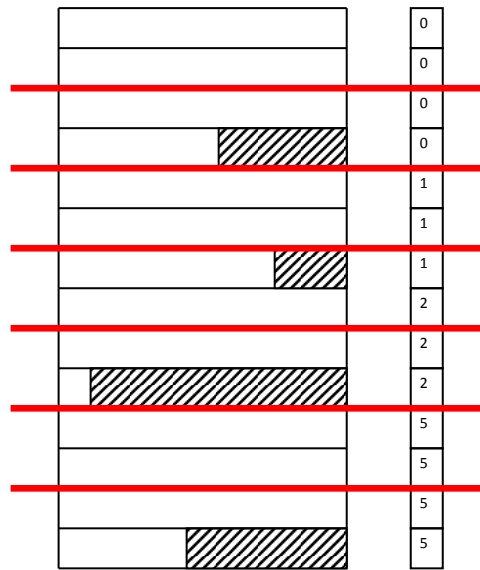


Figure 6.3.: The resulting data from the step in figure 6.1 is shown here. Sequential lines are distributed as workload to one thread block. The limits of the workload are indicated in red in the diagram. The sorted position data is on the left, the associated intensity value for each line on the right. If the intensity values for the fixed image are the same for the complete workload, all the partial histograms can be kept in shared memory. Otherwise the partial histogram has to be written out to memory, when the intensity value changes.

In a final step, the different partial histograms in global memory are then gathered into the final histogram as shown in figure 6.4

### 6.2.3. Reducing Conflicts between Atomic Functions

When the histogram kernel computes partial histograms for the lines described above, the threads of the CUDA kernel need to increment counters, possibly accessing the same location in shared memory from different threads in the same warp. These operations have to be serialized as explained in section 3.6.3. This serialization can have a serious impact on performance. In order to at least reduce the number of collisions, we can add several bins for each counter and add these counters up in a later step. A uniform distribution of counters potentially wastes bins on intensity ranges with only a few values. But the approach by Brosch et al. [188] considers the joint histogram. This has several disadvantages:

1. Computing the joint histogram is computationally more expensive, whereas a 1D histogram can be computed efficiently on the GPU as well. For rigid registration this histogram stays the same throughout the registration and even for deformable registrations it stays relatively accurate unless there are large deformations.
2. The probability distribution is only accurate, if the registration transformation does not change much, compared to its initialization. Even for rigid registrations, this is not the case unless they are well initialized from the very start. For less well aligned images, the joint histogram changes from the original histogram when the registration transformation moves towards the optimum.

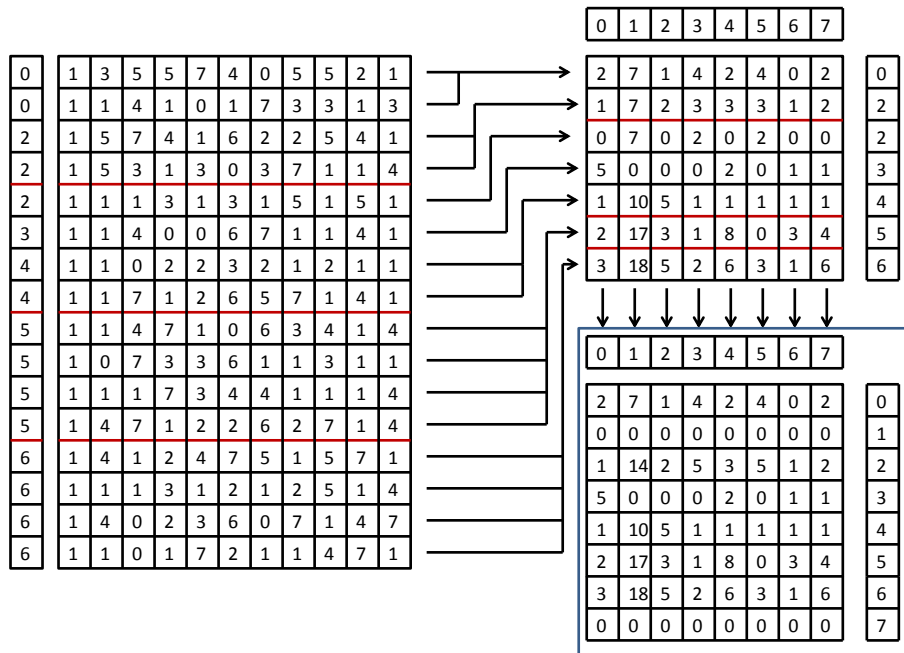


Figure 6.4.: Each block processes its workload and outputs its partial histograms to global memory, whenever the intensity value associated with a line changes. Depending on how many different values a thread block covers, it writes different amounts of global memory. In a final step, these partial histograms are then merged into a final histogram.

Therefore, we compute the intensity distribution of the moving image in a preprocessing step and use this distribution in order to determine the optimal number of bins for each intensity (see figure 6.5). We consider the intensity distribution a probability distribution for collisions between threads updating the same counter in shared memory. If there are more values for one intensity, the number of collisions is expected to rise.

This technique also motivates the choice to only consider one intensity in the fixed volume for each line. If we just consider the amount of shared memory available, histograms for several different intensities of the moving volume can be considered. However, this approach does not leave any memory to add additional counters for common intensity values.

Due to our preprocessing step only the intensities of the moving volume change, therefore, we only need to consider the distribution of intensity values in the moving image.

## 6.3. Results

### 6.3.1. Memory Consumption

After the preprocessing step, the sorted intensity data is not needed fully anymore, only the intensity for each line needs to be kept. Provided we use 256 threads per block, the amount of memory needed is less than 0.5% of the original fixed image. However, aligning the position data to thread block boundaries increases the necessary amount of memory slightly. If  $b$  is the number of bins in the fixed



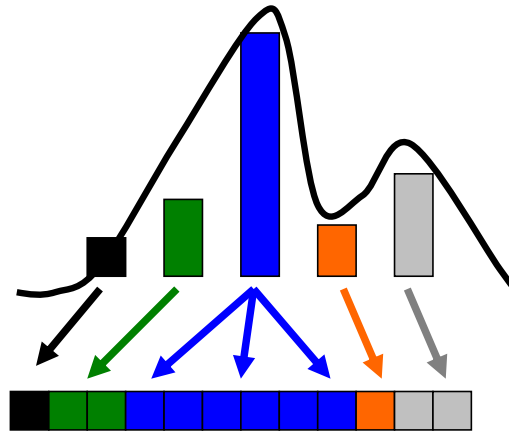


Figure 6.5.: This image shows the distribution of different intensities in the moving image. Since intensities that are more common are more likely to generate collisions between threads the number of bins for the intensities is distributed according to their frequency. Note, however, that intensities that are absent from the moving image cannot necessarily be discarded, since interpolation of the moving image could create these intensities as well. Only intensities below the minimum intensity and above the maximum intensity do not need any bins. The counter used is computed as  $\text{offset}[\text{intensity}] + \text{threadIndex} \% \text{counterNumber}[\text{intensity}]$ .

image, the added amount of memory is smaller than  $b^2$  (in the case of volumes of size  $128^3$ , the memory overhead is about 3 percent of the volume size). On the whole, the additional amount of memory required is slightly more than the amount required for the original fixed image volume.

### 6.3.2. Random Read Patterns and Shared Memory Collisions

Since our algorithm sacrifices coalesced reading patterns for operations on shared memory and coalesced writing of the result, we perform a test on the influence of these different patterns. We also measure the influence of conflicting atomic operations in shared memory on performance. We create several synthetic test cases that compare the timings for completely coalesced reading patterns to completely un-coalesced reading, as well as no collisions in shared memory versus the maximum number of collisions in shared memory. These synthetic data sets illustrate the best case versus worst case performance for the algorithm. Since real medical images do not fall into either of the categories, we also perform these tests on an example of CT images. The results of this experiment are shown in figure 6.6 for an image size of  $128^3$ .

As we can see from this table both a random read access as well as shared memory collisions have a large impact on performance. However, in the presence of shared memory conflicts the impact of a random reading pattern is completely hidden. Only once the collisions are handled the influence of the reading pattern becomes visible again. The timings for the CT images are between the two cases, the reading pattern is not completely arbitrary, but there are also conflicting shared memory accesses.

### 6.3.3. Timings

We have tested our algorithm on an NVidia GTX 285 with 768 GB of memory. For an accurate timing of the processing steps we use CUDA events. Figure 6.7 shows the execution time when computing the

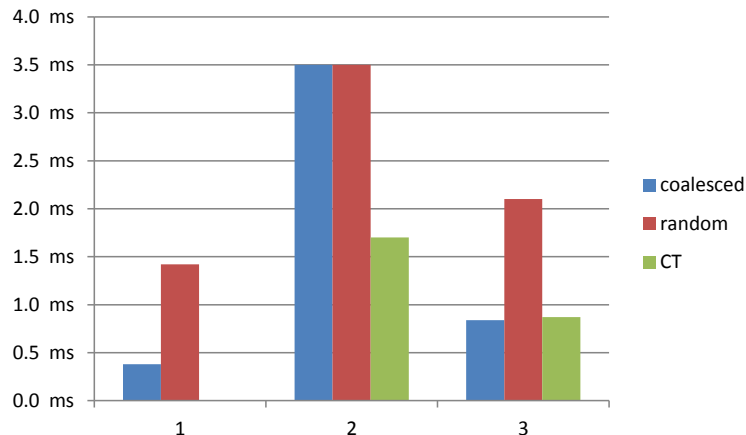


Figure 6.6.: This figure shows different timing experiments for different volumes of the size  $128 \times 128 \times 128$ . Two synthetic patterns result in either completely coalesced reading patterns or a completely uncoalesced reading pattern, whereas the clinical CT data set falls in between these two extremes. In the series 1, there are no collisions in shared memory (this property can only be guaranteed for synthetic patterns, it is not applicable to the CT volume). In the series 2, there are collisions, but they are not explicitly handled, in the series 3, the collisions are handled using the counter distribution.

joint histogram of two CT volumes of different sizes. The size of the volumes is on a logarithmic scale, the size on the x-axis  $n$  indicates an image size of  $2^n$  for both the fixed as well as the moving image. The computation time remains constant for small volume sizes, since the overhead of calling GPU kernels dominates the timing, as it is typical of GPU implementations. But the speed starts to scale linearly with the volume size, once a certain threshold is crossed.

Figure 6.8 displays the relationship between the execution time of the algorithm for a fixed and moving image of size  $128^3$  versus the amount of bins. The algorithm is fairly well behaved and handles different number of bins without much of a difference.

The timings so far only consider the computation of the joint histogram itself, the computation of the mutual information during the registration procedure requires the re-sampling of the moving volume according to the transformation and the computation of the mutual information metric from the joint histogram. The timings for these steps are listed in the table below, once again for a volume size of  $128^3$ .

resample	1.11 ms
histogram	0.9 ms
mutual information	0.22 ms

Compared to the timings presented by Shams et al. [189] of 200 million voxels per second, we process more than 900 million voxels per second (for a volume of  $128^3$  voxels). Even with the slightly faster speed of the NVidia GTX 285 that we used compared to the NVidia GTX 280 that Shams et al. used (1296 MHz processor clock versus 1476 MHz and 141.7 GB/s memory bandwidth versus 151.8 GB/s), our algorithm is still faster by a factor of 4.

We do not perform a separate comparison to CPU algorithms, since previous papers [186, 128] have already established the superior speed of the GPU compared to the CPU, even given suboptimal algorithms.

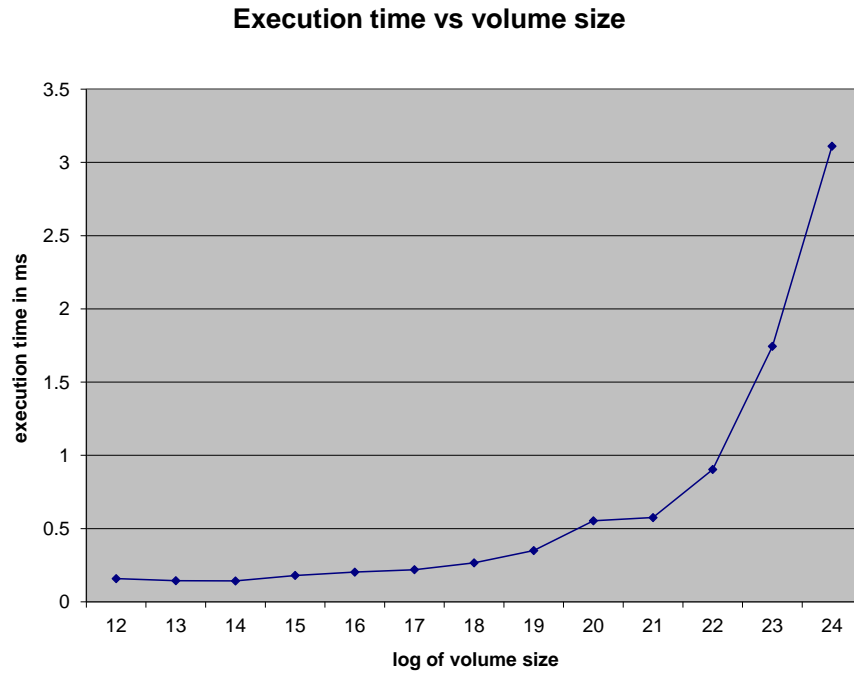


Figure 6.7.: This figure shows the execution time of the histogram kernel (in ms) versus the logarithm of the number of voxels in each image.

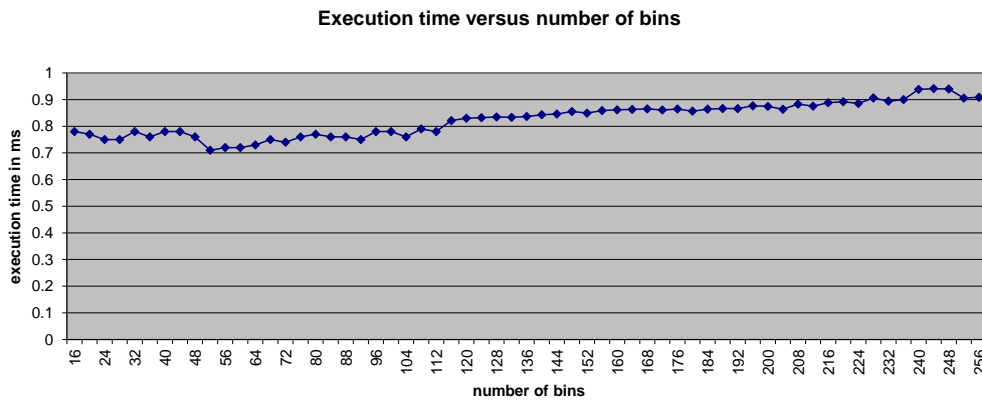


Figure 6.8.: This figure shows the execution time of the histogram kernel (in ms) versus the number of bins for each image. The sizes of the fixed and moving image are  $128^3$  respectively.

Since Sham's algorithm does not need any preprocessing step, we still have to investigate how many iterations have to be performed until the costs are amortized.

The more expensive the preprocessing step is, the more iterations one has to perform before the cost is amortized. For a volume of size  $128^3$ , the preprocessing step takes ca 12.4 ms or as much time as 6 evaluations of the complete mutual information computation. (We do not consider uploading the data

from host memory to the GPU here, since this step is common to all GPU algorithms). After 8 evaluations of the similarity metric, the preprocessing step is amortized compared to the previous best algorithm. With similarity metric evaluations on the order of hundreds per registration, the benefits soon outweigh the costs.

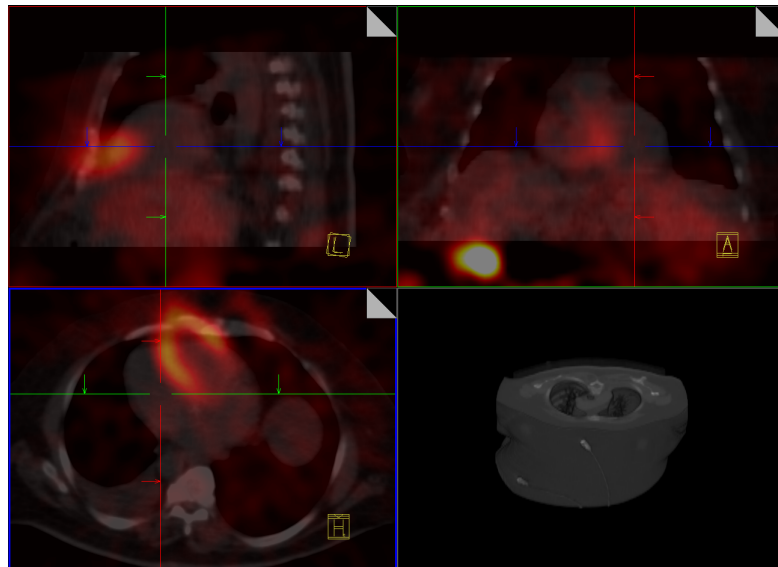


Figure 6.9.: This figure shows an example of the data used for the evaluation of the histogram algorithm. The CT and SPECT volume are clearly unaligned.

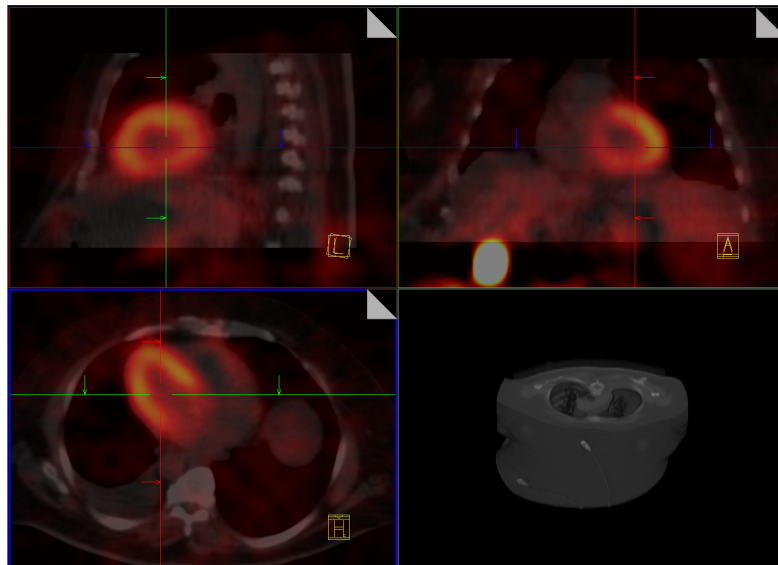


Figure 6.10.: After the registration, the two volumes from figure 6.9 are well aligned.

### 6.3.4. Registration

We demonstrate our algorithm on the registration of a CT with a SPECT volume (see figures 6.9 and 6.10 for an example). We perform rigid registration, but the same algorithm is applicable to deformable registration. The registration is computed on a multi-resolution pyramid of 3 levels with normalized mutual information [40], instead of mutual information. We estimate the probability density function from our sampled representation in the joint histogram using Parzen windowing [35]. The optimization is performed using gradient descent. The free parameters in the optimization are three translational components and three rotation parameters, which results in 12 evaluations of the similarity measure for each registration step (That means that the preprocessing step is amortized within a single registration step). The registration between the SPECT volume of size  $128^3$  and the CT volume with the dimensions  $512 \times 512 \times 38$ , can be computed within 2 seconds.

## 6.4. Conclusion

In this chapter, we have presented a highly optimized version of the joint histogram computation on the GPU, as it is commonly used in important similarity measures like mutual information or the Kullback-Leibler divergence. Our implementation outperforms recent GPU algorithms by a factor of 4. While we sacrifice the ability to perform coalesced reads, our write operations are coalesced and we do not need atomic read-write operations to global memory at all.

We introduce an efficient preprocessing step that amortizes its cost quickly for the typical number of joint histogram computations for registrations. Furthermore, we reduce the impact of conflicting atomic functions in shared memory by adapting the number of bins to the intensity distribution of the moving image. Our algorithm is applicable to rigid as well as deformable registration and yields a fast registration procedure for CT-SPECT registration.



## 7. SPECT Reconstruction on the GPU

This chapter describes the implementation of SPECT reconstruction on the GPU and is based on the paper [5].

### 7.1. Introduction

Computed tomography enables physicians to see inside the patient without invasive surgery. X-Ray CT (often just called CT) provides high-resolution images and is very well suited for imaging structures like bones. Soft tissue can often be better differentiated with MR imaging. Functional imaging on the other hand allows observing metabolic or functional processes that cannot be detected by MR or X-Ray CT. Examples for functional imaging modalities are PET or SPECT. These modalities detect the concentration of radioactive markers in the patient's body. In order to allow the imaging of functional processes, either the radioactive substance needs to be processed by the body in the function that is to be observed, or more commonly, the radioactive marker is combined with a metabolically active substance and indicates the presence of this substance or its derivatives by proxy.

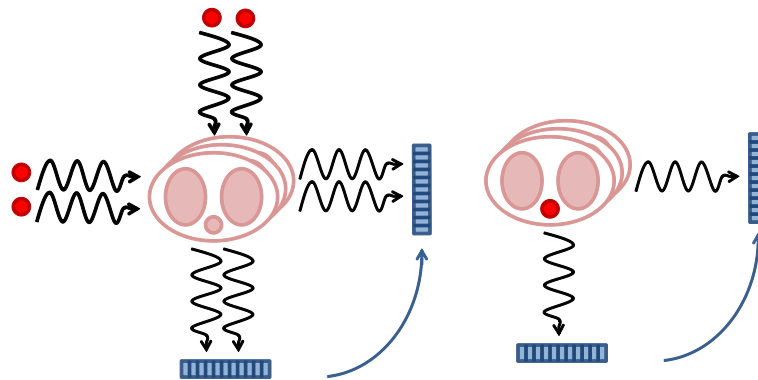


Figure 7.1.: This figure illustrates the difference between transmission tomography (on the left) and emission tomography (on the right). In the case of transmission tomography the radiation source is outside the object that is scanned, in the case of emission tomography, the source is inside.

The procedure works by injecting a radioactive marker into the patient and detecting the emitted photons by a gamma camera. The gamma camera is rotated in a circular trajectory around the patient usually for 360 degrees to ensure a complete field of view and subsequently a good reconstruction. Collimators in front of the detectors ensure that only photons coming at an orthogonal angle hit the detector. This setup provides a two-dimensional projection image for each rotation angle. These projection images are then used as input for the reconstruction algorithm that generates a three-dimensional volume out of the information.

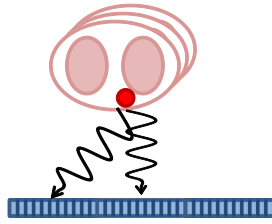


Figure 7.2.: This figure illustrates the setup of a conventional SPECT scanner. Gamma detectors with collimators in front are rotated around a patient. Emitted photons are detected unless they are filtered out by the collimators. In this diagram for example, the photon on the left would be filtered by the collimator, whereas the photon on the right would hit the detector element. These images from the detector for each angle on the circle around the patient are the projection images that are used as input for the reconstruction algorithm.

## 7.2. Method

### 7.2.1. Radon Transform

Mathematically, the process of projection is modeled by the Radon transform. The Radon transform is the integral of a 2D function onto lines, or - in the 3D case - the integral projection of a 3D function onto a plane. For the case of SPECT imaging, the projection images provided by the gamma camera are the Radon transform of the distribution of the radio emitters in the patient.

The solution to reconstructing the 3D volume of the distribution of radio emitters from the emitters is the inverse Radon transform [191, 192]. The filtered back projection is the appropriate analytical solution to a Radon transform without (or at most little) noise. The filtered back projection consists of a filtering step and a back projection step that "smears" the images back into the volume. Commonly, the projection images are filtered first and then the back projection is performed. Filtered back projection is the method of choice for X-Ray CT.

### 7.2.2. Series Expansion Methods

The second option for solving the problem are series expansion methods. In these methods, the solution volume is discretized immediately into voxels and a system of equations that describe these voxels built and solved.

$$Ax = b \tag{7.1}$$

$A$  in the equation above is the system matrix,  $x$  the volume that is to be reconstructed consisting of the voxel  $x_j$ , and  $b$  the vector of observed measurements with each  $b_i$  consisting of the count rate, and a position and orientation in space. In conventional SPECT imaging, the measurement positions and orientations are regular, this however is not the case for freehand SPECT (discussed in chapter 8). The accuracy of the solution depends to a large degree on the system matrix  $A = [a_{ij}]$  that models the contribution of each voxel  $x_j$  to each measurement  $b_i$ . One solution to the system  $Ax = b$  is the maximum likelihood expectation maximization (MLEM) explained in more detail in the following. Note that the system matrix is not necessarily explicitly computed and stored since its size may make this prohibitively expensive.

Due to the characteristics of the SPECT imaging modality, the strong influence of noise as well as the attenuation of the emitted photons by tissue between the emitter and the detector, using the filtered back projection does not result in an accurate reconstruction as is the case for transmission tomography. Instead of a simple Radon transform, the projection images are the result of an



attenuated Radon transform. In this case, filtered back projection (see [193]) provides only an approximate solution though analytical solutions exist as well.

The noise that is typical of processes that rely on radio-active decay adds another problem for the filtered back projection. Therefore, iterative algorithms are used for emission tomography such as Algebraic Reconstruction Technique (ART) [194] or MLEM. Both algorithms start with an estimation of the reconstructed volume that is iteratively refined. This estimated volume is commonly set to 1.0, but additional prior knowledge can be used to provide a more appropriate initialization. This estimation is then iteratively updated according to how accurately the reconstructed volume explains the actual measurements. This is done by first forward projecting the estimation volume onto the projection images. Correction factors are then computed by comparing the forward projected images to the actual projection images. The correction images are then back projected into the estimation volume.

### 7.2.3. Maximum Likelihood Expectation Maximization

In the following, we discuss the Maximum Likelihood Expectation Maximization since our implementation is based on this algorithm. MLEM has been explained and named by Dempster [195] and popularized for emission tomography by Shepp [196]. MLEM differs from ART in the way the correction image is computed. ART uses the difference between the original projection image and the projection image forward projected from the estimation volume, whereas MLEM uses the division of the two.

The MLEM algorithm can be divided into several different components. We use the following notation in this section. The incoming projection images are the set  $P$  of pixels  $p_i$ :  $P = p_i$  with  $|P|$  the number of pixels (for now the distribution of the individual pixels into different projection images is ignored). The output is the reconstructed volume  $V = v_i$  with  $v_i$  the individual voxels and  $|V|$  the total number of voxels. A matrix  $W$  of elements  $w_{ij}$  models the contribution of voxel  $v_j$  to the pixel  $p_i$ . In physics terms the weight  $w_{ij}$  gives the probability that a photon emitted from voxel  $v_j$  is detected by the gamma camera at position corresponding to pixel  $p_i$ . The physical properties of the detector and emitter determine these probabilities.

The MLEM algorithm can be described in the following formula with  $v_i^{(k)}$  the estimate for voxel  $v_i$  in the  $k$ -th iteration.

$$v_j^{(k+1)} = v_j^{(k)} \frac{\sum_{i=1}^{|P|} \left( \frac{p_i}{\sum_{m=1}^{|V|} w_{im} v_m^{(k)}} w_{ij} \right)}{\sum_{i=1}^{|P|} w_{ij}} \quad (7.2)$$

In the following, we will show how this procedure can be decomposed into several basic operations.

#### Forward Projection

The first of these operations is the forward projection. The forward projection operation sums up the contribution of each voxel according to its weight toward the pixel  $p'$ . The weight (the contribution of a voxel to a particular pixel) can be computed by sampling the reconstruction kernel of that voxel along the ray through that pixel.

$$p'_i = \sum_{m=1}^{|V|} w_{im} v_m \quad (7.3)$$

The forward projection image is then compared to the actual projections by computing a correction image in the next step.

### Correction Images

This correction image is used for adjusting the estimated solution. It consists of the element wise division of the observed pixel value in the projection images by the value at the same pixel location resulting from the forward projection step (that computed the value  $p'_i$ ).

$$c_i = \frac{p_i}{p'_i} \quad (7.4)$$

### Backprojection

The process of back projection then performs the projection of the correction image back into the volume. In mathematical terms, this process can be described as the following:

$$v_j = \sum_{i=1}^{|V|} p_i w_{ij} \quad (7.5)$$

For our purposes, the pixels  $c_j$  of the correction image are smeared into the volume.

$$s_j = \sum_{i=1}^{|P|} c_i w_{ij} \quad (7.6)$$

Note that there is no scaling according to the number of pixels that are back projected into the volume. This is done in the next step.

### Volume Update

In the volume update the estimated solution volume is updated with the volume created from the correction images. The normalization step missing from the back projection is performed here.

$$v_j^{(k+1)} = v_j^{(k)} \frac{s_j}{\sum_{i=1}^{|P|} w_{ij}} \quad (7.7)$$

An overview over the complete process can be found in image 7.3.

#### 7.2.4. Ordered Subset Expectation Maximization

Computing the scaling factors in the Expectation Maximization is expensive because the projection images for every projection angle have to be forward projected from the estimated solution volume. Only then can the correction images be computed and back projected into the volume. This limits the number of correction steps that can be performed (or requires a long computation time alternatively). Ordered Subset Expectation Maximization (OSEM) [197] computes the correction images more often and thus improves the convergence rate by forward projecting only a subset of all the projection images. If we are considering image 7.3 again, the steps performed for the OSEM algorithm are the same as for the MLEM algorithm, with the only difference that only a subset of projection images  $|P|$  is used and not the complete set.

If  $I_1, \dots, I_N$  are the projection images corresponding to the projection angles  $\alpha_1, \dots, \alpha_N$ , then these images are divided into  $M$  subsets. The first subset of projection images consists of  $I_1, I_{M+1}, I_{2M+1}, \dots$ , the second set consists of  $I_2, I_{M+2}, I_{2M+2}, \dots$  and so on. The volume update step is now performed for each of these subsets, instead of performing it once for all the projection angles. This speeds up the convergence of the algorithm considerably.

Let  $P_i$  be the set of pixel indices for the subset  $I_i$ . Then the equation 7.3 for the the forward projection becomes:

$$v_j^{(k+1)} = v_j^{(k)} \frac{\sum_{i=1}^{|P|} \left( \sum_{m=1}^{|V|} w_{im} v_m^{(k)} \right) p_i w_{ij}}{\sum_{i=1}^{|P|} w_{ij}}$$

$p'_i = \sum_{m=1}^{|V|} w_{im} v_m$       forward projection  
 $c_i = \frac{p_i}{p'_i}$       correction image  
 $s_j = \sum_{i=1}^{|P|} c_i w_{ij}$       back projection  
 $v_j^{(k+1)} = v_j^{(k)} \frac{s_j}{\sum_{i=1}^{|P|} w_{ij}}$       volume update

Figure 7.3.: This figure visualizes the breakdown of the Maximum Likelihood Expectation Maximization (MLEM) into its different components. The equation on the top describes a complete update of the estimated solution from one iteration to the next iteration. The equation in the blue box describes the forward projection part and where it fits in the general framework. The equation for computing correction images is marked by the red box and the back projection part is marked by the yellow box.

$$s_j = \sum_{i \in P_i} c_i w_{ij} \quad (7.8)$$

and the equation 7.2 for the maximum likelihood expectation maximization becomes:

$$v_j^{(k+1)} = v_j^{(k)} \frac{\sum_{i \in P_i} \left( \frac{p_i}{\sum_{m=1}^{|V|} w_{im} v_m^{(k)}} w_{ij} \right)}{\sum_{i \in P_i} w_{ij}} \quad (7.9)$$

Using these equations, the algorithm is performed until it converges or for a preset number of iterations. Employing a single subset for the OSEM algorithm results in the standard EM algorithm.

### 7.2.5. Attenuation Correction

Photons emitted by the tracer have a tissue-dependent probability to reach the detector. Bone tissue for example attenuates photons much more strongly than soft tissue. Neglecting this effect can lead to inaccurate reconstruction results. CT volumes are a good way to provide this information. Modern CT/SPECT scanners provide a CT volume co-registered to the SPECT scan that not only provides the attenuation correction, but also additional anatomical information that cannot be gleaned from the SPECT scan alone. However, this information comes at the cost of a higher radiation dose. In order to include the correct attenuation, the attenuation factor  $A_j$  for voxel  $v_j$  has to be computed. The attenuation factor depends on the attenuation coefficient  $M_j$ .  $A_j$  is computed by sampling the volume of coefficient factors  $M_j$  using the reconstruction kernel of the attenuation coefficient  $\mu_j$ . This computation results in the following formula.

$$A_j = e^{-(M_j)} \quad (7.10)$$

Attenuation correction can be modeled in both forward as well as backward projection, though for efficiency reasons, the attenuation could only be modeled in the forward projection step [198]. With attenuation correction included, the formula for forward projection

$$v_j = \sum_{i=1}^{|V|} p_i w_{ij} \quad (7.11)$$

becomes the following:

$$v_j = \sum_{i=1}^{|V|} p_i \left( \prod_{k=1}^n A_{S_k} \right) w_{ij} \quad (7.12)$$

The set of attenuation factors  $A_{S_j}$  are the factors corresponding to the sample points  $S_1, S_2, \dots, S_n$  that sample the volume of attenuation coefficients along the ray through pixel  $p_i$ . And the equation for back projection

$$p'_i = \sum_{m=1}^{|V|} v_m w_{im} \quad (7.13)$$

becomes the following:

$$p'_i = \sum_{m=1}^{|V|} v_m \left( \prod_{k=1}^n A_{S_k} \right) w_{im} \quad (7.14)$$

Implementing the projection algorithm is a well-investigated problem in volume visualization research with many different algorithms to choose from for example shear-warp [199], splatting [200], 3D texture slicing [135] or ray casting [201].

### 7.2.6. The Point Spread Function

Both forward as well as backward projection in the case of SPECT imaging are complicated by the SPECT point spread function (PSF) [202].

While collimators restrict the angle of the detected photons, they cannot ensure a perfectly orthogonal angle, instead photons from within a certain angle are detected. This results in a resolution that diminishes with increasing distance of the radiation source from the detector. Or in the term of a PSF, a point source is registered by an increasing number of detectors with increasing distance. For a more accurate reconstruction, this effect has to be taken into account as we will see in the following sections.

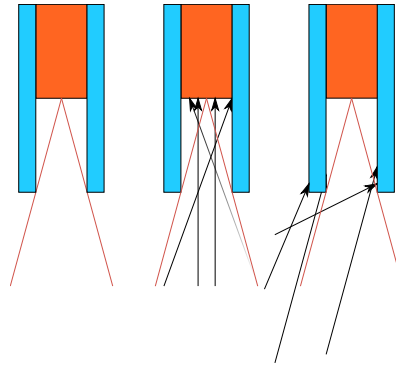


Figure 7.4.: This diagram illustrates the problem of the point spread function in SPECT imaging. The collimator (blue) prevents photons (black arrows) from reaching the detector element (orange) if they do not arrive at an orthogonal angle. This filter does not work perfectly. Depending on the size of the collimator a wider angle of photons is detected. (Collimators are used because it is not possible to use an equivalent to lenses as for visible light). The left side of the diagram shows the detector element and its field of view, the middle shows examples of photons that are registered by the detector. With increasing distance, a larger area is covered, resulting in a decreased resolution. The right side of the diagram shows examples of photons that are filtered out. Note that more sophisticated models are possible, for example modeling the likelihood of penetration through the collimator. This detailed modeling is costly, especially for the amount of data generated by conventional SPECT, but is manageable and improves reconstruction in the case of freehand SPECT, see chapter 8.

## 7.3. GPU Implementation

### 7.3.1. Previous Work

Since we have covered GPGPU already in chapter 3.6, we will concentrate on volume reconstruction on the GPU in the following. In 1994, Cabral [135] already suggested using graphics hardware in order to accelerate reconstruction. In this case, the resampling that is necessary for the filtered backprojection is implemented as texture resampling operation which is hardware-accelerated on the Reality Engine trademark system used in this paper. Mueller et al. [203] use the SGI Octane in 2000 for the re-sampling in the SAR algorithm. Working around the limited number of bits available per channel in the textures and the accumulation buffer is a major part of this paper.

But only with the emergence of mass market programmable graphics cards did this research topic become more active. Chidlow et al. [204] implement the expectation maximization algorithm on cards of the NVidia GeForce 4 generation. But before the availability of 32-bit floating point textures let alone 32-bit integer or even double support, working around the precision limits of the hardware is once again a major problem, as was the case in the previous effort of Mueller et al. [203].

Wang et al. [205] extend Chidlow's work by incorporating the effect of the point spread function on a per-voxel base. The point spread function is modeled on a per-voxel base, but by exploiting several symmetries in the matrix, the amount of data is kept to a manageable amount. Later research performs MR reconstruction [92] or cone beam reconstruction [206, 207] on the GPU.

### 7.3.2. Implementation

The ordered subset expectation maximization algorithm is implemented on the GPU using C++ and OpenGL. The expected reconstruction volume is initialized with a cylindrical shape. The main algorithm consists of three nested loops. The outermost loop iterates a fixed amount of time (in our tests 10 times). The loop below iterates over the different subsets, and the innermost loop iterates over the projection angles within the current subset (see figure 7.5).

In the following, we discuss modeling the point spread function and the attenuation first, since they influence the implementation of the forward and backward projection.

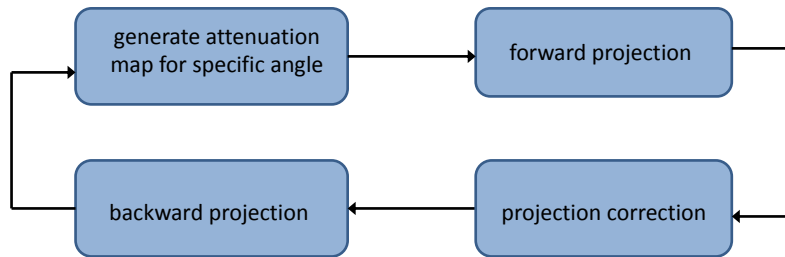


Figure 7.5.: This flow diagram illustrates the inner loop of the GPU-based reconstruction pipeline. The loop is performed for each angle. The attenuation map is generated (or fetched from a cache) in order to use it for both the forward as well as the backward projection. After the forward projection, the projection correction is performed and the correction images are back projected into the correction volume.

### 7.3.3. Modeling the Point Spread Function

In order to accurately replicate the physical response of the collimator, the PSF models are often 3D, i.e. they are depth dependent with the extent of the modeling function widening both axially and trans-axially as the distance of the object from the collimator increases. Modeling the PSF matrix is computationally prohibitive both in terms of memory requirements and computation time. Therefore the PSF is modeled as blurring that is dependent on the distance of the slice. Instead of recomputing the Gaussian kernel for every distance, the volume can be divided into different slabs [208] that share the same convolution kernel. Depending on how many slices are contained per zone, the approximation is more or less accurate.

### 7.3.4. Attenuation Modelling

The attenuation volume is generated by a co-registered CT volume, by marching along the ray of emission and accumulating the value. The attenuation volume is different for each angle, so attenuation correction becomes an expensive operation. In the case of the forward projection, the attenuation volume can be generated in the same step, as the voxel are forward projected, the values in the CT volume can be accumulated. Since the backprojection is voxel-driven, this is not possible in this case though. This problem is an instantiation of the gather versus scatter approach to GPU programming. Using OpenGL, the operation has to be performed in a gathering approach. Three options on how to handle the attenuation volume exist, depending on hardware constraints. The attenuation volume can always be generated on-the-fly, trading computing time for lower memory demand, the attenuation volume for each angle can be generated exactly once and then be cached, trading higher memory

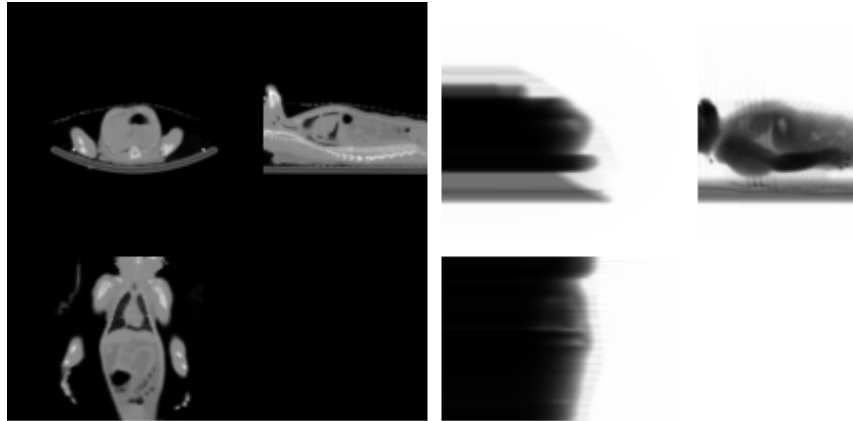


Figure 7.6.: The left side of this figure shows three views of the CT volume that is used for computing the attenuation map. The right side shows the resulting attenuation map for a specific angle. Note how the volume is accumulated in one direction.

consumption for lower computing power, or only a certain number of attenuation volumes is cached depending on a fixed memory budget. This hybrid approach allows reducing the computing time required as much as possible, while using all the available memory for caching.

### 7.3.5. Forward Projection

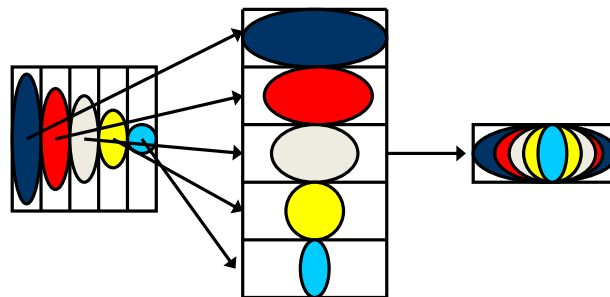


Figure 7.7.: This image illustrates the process of forward projection. The forward projection is performed for each slab of the volume into a different 2D region of the target texture. The resulting 2D images are then convolved with a Gaussian kernel added in a final compositing step.

The forward projection (see figure 7.7) performs a rotation of the volume first using bilinear interpolation (bilinear interpolation instead of trilinear interpolation can be used in this case, since the projection images are acquired around the z-axis of the reconstruction volume). The projection step is performed in one pass, but the voxels are projected into different regions of the target texture according to the slab they are in. The different zones in the target texture are then convolved with different Gaussian kernels according to the depth of the originating slab in order to model the point spread

function approximately. The blurred projections are then composited into the final projection image.

### 7.3.6. Projection Correction

The correction of the projection images is performed by dividing the original projection images by the result of the forward projection on a pixel-by-pixel basis. This can be implemented in a straightforward way on GPUs. The only drawback is that the projection images are relatively small, so this step does not fully exploit the capabilities of the GPU that are a better match for larger images.

### 7.3.7. Backward Projection

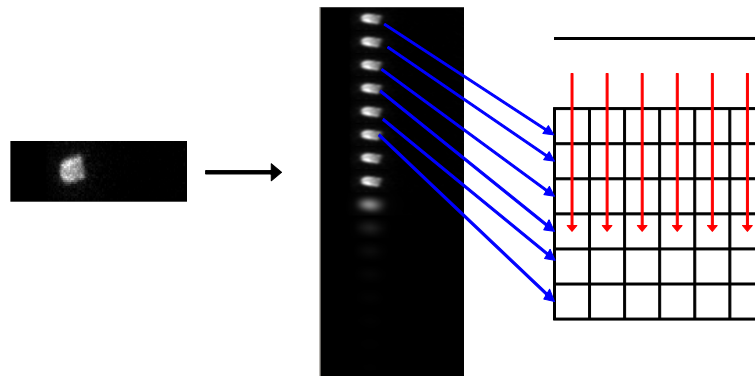


Figure 7.8.: The backprojection is performed in analogy to the forward projection. The projection image for the current angle is copied and then convolved with a Gaussian kernel for each slab in the volume. Then during back projection the lookup for each slab in the result volume is performed into the appropriately convolved image.

In the case of the backward projection (see figure 7.8), the projection images are smeared across the volume. The implementation works voxel-driven, so each voxel asks by which pixel in the projection image it is affected. Once again, this process is somewhat more complicated because of the PSF. In the first pass, the smoothed projection images are generated, with the specific parameters for each zone. Just like in the forward projection, one texture holds the images for all the zones. Now there are two algorithmic approaches to the backward projection. The first one projects directly. For each element in the volume, the rotated position is determined, which is used to perform the lookup into the texture that holds the blurred images. Depending on the depth, a different zone is chosen. If this approach is used with attenuation correction, the attenuation volume before the rotation back needs to be used. This volume is generated anyway, but if this operation is not performed on the fly, the amount of memory needed is doubled. In the second approach, the blurred images are projected orthographically into the volume, and then a rotation of the volume is performed.

The estimated reconstruction result is updated at once for the whole subset. For normalizing the result, the average transmission map is used which is being kept track of during the iteration over the projection angles of the subset.

### 7.3.8. Volume Update

The volume update operation updates the volume with the new back projected volume. In the case of attenuation correction the final step in computing the average transmission map is performed here as well, namely scaling the sum of the attenuation volumes and inverting the result. When subsets are



used, several back projected volumes are created before the update volume step is performed, the back projection shader adds the previous back projected volume to the current back projected volume.

### 7.3.9. Convolution

The convolution of the images - necessary to model the point spread function - is implemented as a two-pass approach. Since the convolution is separable, rows and columns are convolved separately in order to reduce the number of texture fetches. For the forward projection and the backward projection, the convolution creates an atlas of images smoothed with different blurring kernels. This approach allows reducing the number of passes. The image 7.7 shows the creation of this atlas for the forward projection. In the case of the forward projection, the forward projection of each zone is projected to a different position in the atlas texture. This means that for the convolution in the vertical direction the shader performs a check to decide whether the texture fetch crosses the border between the images for the different zones. In this case, the texture coordinate is clamped. This clamping does not have to be performed for the backward projection, because the vertical convolution is performed first during the creation of the texture atlas (so the hardware takes care of the clamping) and the vertical convolution pass does not access the wrong region, since the images for the different zones are stacked in the horizontal direction.

## 7.4. Results

The following speed was achieved using a CPU Xeon 5150 Dual Core, Dual Processor and a single-GPU NVidia Quadro FX 4500. The reconstruction algorithm uses 10 iterations and divides the volume into 16 different depth zones. 64 different views are the input to the reconstruction algorithm. The timings are given in seconds for the whole reconstruction. This includes the transfer of the input data to video memory and the readback of the final result of the reconstruction to main memory. The software version is a highly optimized implementation that makes full use of all processor cores. Both versions operate on 32-bit floating point data.

	without attenuation correction	with attenuation correction
volume size:	128 x 128 x 128	128 x 128 x 54
CPU Time:	7.25 s	7.71 s
GPU Time:	7.1 s	12.3 s

The GPU timings, however, improve by a factor of about 2, using the more recent NVidia Geforce 8800 GTX. In this case hardware supported floating point blending is used, whereas versions that run on previous generations implement this explicitly in the shader.

	without attenuation correction	with attenuation correction
1. volume (128 x 128 x 82)	2.1 s	4.65 s
2. volume (128 x 128 x 90)	2.24 s	4.73 s
3. volume (128 x 128 x 90)	2.24 s	4.5 s

## 7.5. Conclusion

In this chapter, we have demonstrated a reconstruction pipeline with the following features:

1. The entire reconstruction pipeline is implemented on the GPU without bus transfer between main memory and video memory.

## 7. SPECT Reconstruction on the GPU

---

2. The algorithm uses an optimized blurring step on the GPU to model the point-spread-function in order to improve spatial resolution in the reconstruction images.
3. The convolution of several layers that are generated during the forward projection and backward projection is combined into one pass, in order to avoid CPU interaction.

The GPU implementation performs considerably faster than a state-of-the-art, optimized CPU implementation.

## 8. 1D–3D Registration for Intra–operative Nuclear Imaging in Radio–guided Surgery

3D functional nuclear imaging modalities like SPECT or PET provide valuable information, as small structures can be marked with radioactive tracers to be localized before surgery. This positional information is valuable during surgery as well, for example when locating potentially cancerous lymph nodes in the case of breast cancer. However, the volumetric information provided by pre–operative SPECT scans loses validity quickly due to posture changes and manipulation of the soft tissue during surgery. During the intervention, the surgeon has to rely on the acoustic feedback provided by handheld gamma–detectors in order to localize the marked structures.

In this paper, we present a method that allows updating the pre–operative image with a very limited number of tracked readings. A previously acquired 3D functional volume serves as prior knowledge and a limited number of new 1D detector readings is used in order to update the prior knowledge. This update is performed by a 1D–3D registration algorithm that registers the volume to the detector readings. This enables the rapid update of the visual guidance provided to the surgeon during a radio–guided surgery without slowing down the surgical workflow. We evaluate the performance of this approach using Monte–Carlo simulations, phantom experiments and patient data, resulting in a positional error of less than 8 mm which is acceptable for surgery. The 1D–3D registration is also compared to a volumetric reconstruction using the tracked detector measurements without taking prior information into account, and achieves a comparable accuracy with significantly less measurements.

### 8.1. Introduction

Functional nuclear imaging is routinely used for diagnostic purposes as well as for therapy planning. It allows the localization of very small structures, but this information cannot be easily exploited in the operating room without intra–operative nuclear imaging capabilities that do not impede the surgical workflow, for example in the case of breast cancer treatment.

#### 8.1.1. Clinical Problem

Sentinel lymph node biopsy (SLNB) is an important example of a procedure that uses radioactive tracers in order to mark small anatomical structures [209, 210], in this case lymph nodes. These tracers are molecules in which at least one atom has been replaced by a radio–isotope. This results in molecules that are processed equivalently by the human metabolism, but allow tracing their location and the location of possible derived products by the radiation they are emitting (e.g. gamma rays for SPECT). The concentration and location of these tracers is then visualized using functional imaging, for example SPECT.

Sentinel lymph node biopsies are the standard of care in early stage breast cancer treatment due to low morbidity and shorter hospital stays [211, 212]. In cases of breast cancer, metastases typically occur in the lymphatic system. The sentinel lymph node is the first lymph node that the lymph fluid in a certain region drains to, and therefore the first lymph node potentially invaded by cancerous cells originating from the tumor site. If the sentinel lymph node does not show signs of cancer, it is highly probable that the cancer has not spread any further.

In order to identify the sentinel lymph node, a low–energy radioactive tracer is injected near the site of the tumor. Following the flow of the lymph fluid, the tracer aggregates in the sentinel lymph node first.

This procedure that identifies to which lymph nodes the lymphatic fluid from the tumor site drains to is called lymphatic mapping, and needs to be done for surgery planning as well as intra-operatively in order to excise the correct sentinel lymph node. A pre-operative 2D scintigraphy is routinely performed for interventional planning beforehand.

During surgery, a handheld gamma detector (working like a one-pixel camera) records the amount of radiation, in the form of a count rate [12]. The count rate (or the number of detected nuclear events) constitutes a 1D signal. This 1D signal is converted into an acoustic feedback (functioning like a Geiger counter), with the number of audible clicks proportional to the count rate. The acoustic feedback guides the surgeon to the location of the radioactively labeled sentinel lymph node. In most settings, the surgeon relies on the acoustic feedback to locate the position of the lymph node. The sentinel lymph node is then excised and examined histologically. The guidance by the gamma detector allows targeting the sentinel lymph nodes explicitly instead of excising all the lymph nodes in the drainage area.

A pre-operative SPECT scan can be used in order to provide additional visual guidance for the surgeon during the procedure, for example if the 2D scintigraphy alone is not sufficient for lymphatic mapping due to the relative positions of the lymph nodes and the injection site. However, the lymph nodes change their position with the deformation of embedding tissue due to posture changes or intra-operative manipulation of the tissue, resulting in displacement up to a few centimeters. This forces the surgeon to perform a mental update of the provided imaging information to the current situation according to acoustic feedback from the gamma detector [213]. An updated visual guidance would relieve the surgeon from this task and lead to a faster and more accurate location and excision of the lymph nodes.

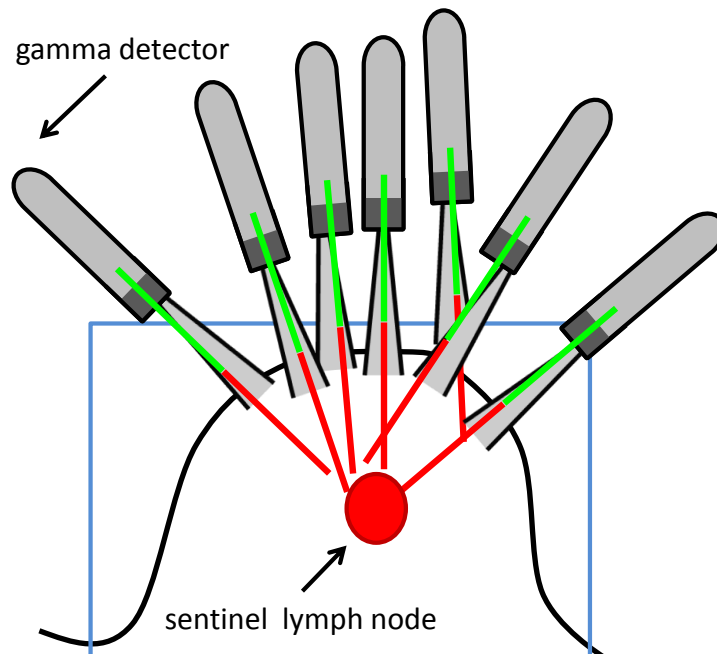


Figure 8.1.: This figure visualizes a freehand scan as they are created intra-operatively. The lines in blue indicate the region of interest, the sphere in red indicates a radioactively marked lymph node. The gradient lines from red to green indicate the orientation of the measurements.

### 8.1.2. Intra-operative Guidance and Interventional Imaging

During interventions, surgeons might use pre- or intra-operative imaging for further guidance. Using intra-operative imaging has the advantage that the information is up-to-date and has not lost some of its value due to changes in posture, tissue or other effects. On the other hand, intra-operative imaging often provides lower resolution images or a limited dimensionality compared to pre-operative imaging. Fusing intra- and pre-operative imaging information is therefore an active research topic. Radiation therapy is one example where CT data is registered to 2D X-Ray images, in order to deliver radiation precisely to cancerous tissue without affecting surrounding tissue. Also in radiation therapy, diagnostic ultrasound data is fused with treatment planning CT data [214]. Other examples include the registration of intra-operative ultrasound data with pre-operative MRI data in order to correct for intra-operative brain shift [215].

Freehand SPECT (fh-SPECT in the following) is a recently introduced approach to intra-operative nuclear imaging [216, 217, 218, 219]. It provides 3D visual information beyond the acoustic feedback of the handheld gamma detector.

In contrast to conventional SPECT imaging with 2D gamma detectors in a fixed, optimized geometric configuration, fh-SPECT uses spatially tracked 1D gamma detectors with position and orientation information. In order to get visual guidance, the surgeon moves the gamma detector in her/his hand, manually covering the region of interest. This results in a "freehand" geometry of detector measurements (see figure 8.1 for an example) that is ad-hoc, non-uniform, sparse and has only a limited angular coverage, compared to the fixed, fully sampled detector geometry in a conventional SPECT scanner. These measurements are then used to reconstruct a 3D radioactivity distribution within the volume of interest. Compared to conventional SPECT, the flexible positioning allows moving the detector much closer to the area of interest and therefore the source of the radiation. This compensates for the lower detection statistics, with only a few hundred thousand events compared to the billions of events in a typical SPECT scan.

fh-SPECT has been introduced as a prototype in 2007 [220]. A validation study published in 2010 [221] performs a comparison between fh-SPECT reconstructions with SPECT/CT for lymphatic mapping. Since then, an fh-SPECT implementation – "declipse SPECT" – is available commercially provided by SurgicEye GmbH (Munich, Germany).

fh-SPECT allows the intra-operative reconstruction of SPECT volumes and thereby provides visual guidance to the surgeon, compared to the purely acoustic guidance when using the gamma detector like a Geiger counter. However, even the fh-SPECT setup requires a dedicated manual scanning process of the region of interest for a duration of 2 to 5 minutes.

Because of this, the tissue movement during surgery remains an issue even with fh-SPECT, since time constraints prevent the surgeon from repeating the image guidance as often as necessary. Our previous work [6] follows an approach more in line with the previously mentioned references on registration for intra-operative guidance. Instead of performing a direct reconstruction from the detector measurements, this approach uses information from a previously acquired SPECT scan and uses the tracked detector measurements in order to register this SPECT scan to the intra-operatively acquired detector measurements to provide updated visual guidance. The set of count rates (1D) of the freehand scan are registered to count rates simulated from the 3D SPECT volume, using the auxiliary information from the tracking device recording the position and orientation of the gamma detector. This approach needs only a very limited number of measurements resulting in short (<30 s) exploratory sweeps of the region of interest using the handheld gamma detector, and effectively provides a fast reconstruction of the area of interest using prior information and registration algorithms. In this paper, we expand on our previous work [6], adding a more accurate modeling of the physical properties for the gamma detector, as well as experiments using Monte Carlo simulations, phantom experiments and experiments with clinical data. In the following we explain the technical background of our approach, detail the algorithms used to perform the registration, and demonstrate the effectiveness of our approach on an extensive set of experiments.

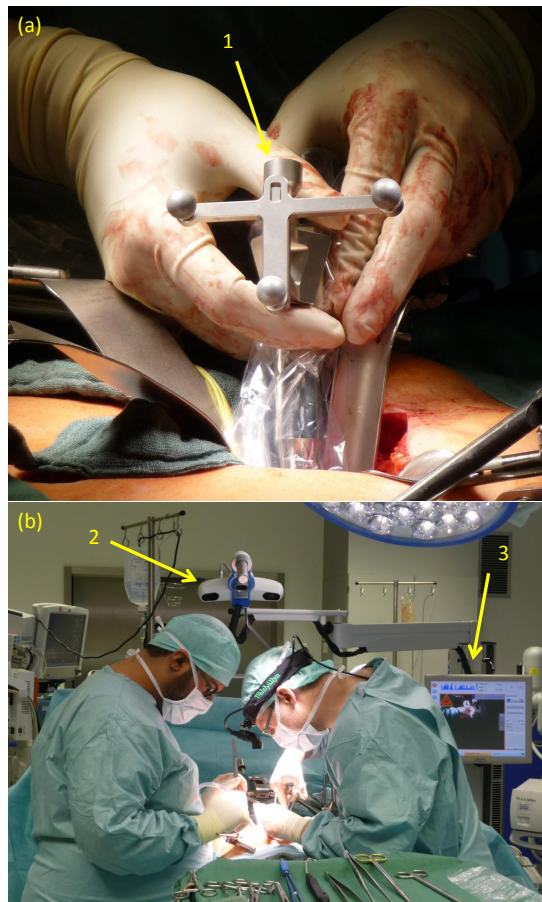


Figure 8.2.: This figure shows an exemplary fh-SPECT setup: (a) shows the surgeon holding the gamma detector with the markers for optical tracking (1), (b) shows an overview during a procedure with the tracking device (2) and the monitor (3) in the background showing an augmented reality view of the surgery site.

## 8.2. Methods

### 8.2.1. Surgical Workflow

Figure 8.2 shows an example situation in the freehand SPECT setup. Figure 8.2 (a) shows the gamma detector with the attached markers for optical tracking, whereas (b) gives an overview during a surgery. An optical tracking system as shown in figure 8.2 (b) tracks the target attached to the gamma detector. Using the tracking system and a rigidly attached and calibrated video camera, the surgeon is provided with an augmented reality view of the surgery site on the monitor attached to the workstation (3), showing the field of view of the gamma detector as well as the reconstructed volume overlaid on top of the video feed of the camera.

Using the augmented reality view, the surgeon can then locate the lymph nodes. In case of changes or deformations of tissue, for example due to surgical manipulations, the relevant lymph node then has to be re-located. Starting at the old location of the lymph node as indicated by the visual guidance, the surgeon performs exploratory gyrating movements with the gamma detector, following the acoustic feedback to the current location of the radioactively labeled lymph node. Our approach uses the data (position, orientation and readout of the gamma detector) recorded during this exploratory sweep to

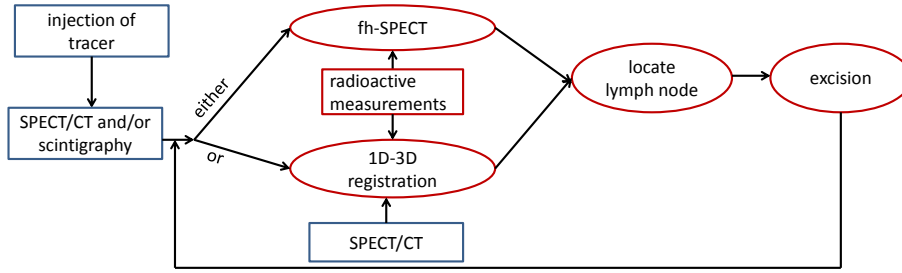


Figure 8.3.: This figure shows the surgical workflow for the SLNB procedure using fh-SPECT or the proposed 1D-3D registration with the pre-operative steps in blue and the intra-operative steps in red (see section 8.2.7).

perform 1D-3D registration to update the visual guidance online. This way, the visual guidance is updated when necessary, without having to perform a full new fh-SPECT acquisition and reconstruction, which would take several minutes and would interrupt the workflow. Figure 8.3 gives an overview over the surgical workflow using fh-SPECT.

### 8.2.2. Input Data and Output

In order to perform an update of the visual guidance, our method requires a previously acquired input volume  $V \subset \mathbb{R}^3$  as prior information, for example from pre-operative SPECT or intra-operative fh-SPECT, as well as a series of tracked 1D detector measurements. The tracked detector measurements  $B = \{b_j = (c_j, p_j, o_j) | 0 \leq j < k\}$  consist of the count rate  $c_j$  (counts per second) of the detector, and the synchronized position and orientation  $(p_j, o_j) \in \mathbb{R}^3 \times \mathbb{R}^3$  for each  $b_j$  as acquired by the tracking device. The output of the method is an updated 3D volume  $\hat{V} = \hat{T}(V)$ , where  $\hat{T} : \mathbb{R}^3 \rightarrow \mathbb{R}^3$  is an approximation of the optimal transformation transforming the no longer valid volume  $V$  into the desired volume  $\hat{V}$ .

### 8.2.3. Registration Procedure

The updated 3D volume  $\hat{T}(V)$  is generated by 1D-3D registration. For the purposes of the registration algorithm, the set of tracked counts  $c = \{c_0, \dots, c_k\}$  is the *fixed image*, while the *moving image* is the set of count rates  $c' = \{c'_0, \dots, c'_k\}$  of the set of measurements  $B' = \{(c'_j, p_j, o_j) | 0 \leq j < k\}$ , simulated from the current transformed volume  $T(V)$ , where  $T : \mathbb{R}^3 \rightarrow \mathbb{R}^3$  denotes the current estimate of  $\hat{T}$ . The similarity measure employed is sum of squared differences (SSD). This results in the minimization of

$$\hat{T} = \operatorname{argmin}_T \sum_{j=1}^k (P_j(T(V)) - c_j)^2, \quad (8.1)$$

where  $P_j(T(V)) = c'_j$  denotes the  $j$ -th simulated measurement from  $T(V)$ . The optimization problem in equation (8.1) can be minimized with standard optimization techniques, we use gradient descent. Typically, only a small part of the region that contains the current node of interest is covered by measurements. If the original position of the node is only covered by very few measurements, we use a heuristic that initializes the start position for the registration. We take the reading with the highest count rate (as this is close to the new position of the lymph node) and intersect the line defined by the

scanning direction and the position of this measurement with the plane orthogonal to the scanning direction that contains the original position of the lymph node.

#### 8.2.4. Detector Model

The most crucial step in the equation (8.1) is the projection step which performs the mapping  $P_j : T(V) \mapsto \mathbb{R}$ . This projection simulates the *moving image* from the transformed input volume  $V$  and maps the 3D volume onto 1D measurements  $c_j$ . We use the projection model introduced by Hartl et al. [222] for a good compromise between physical accuracy and computational demands.

Radiation sources in  $T(V)$  are modeled as a Poisson distribution dependent on the activity. We discretize  $T(V) = (x_i)_{i=1,\dots,n}$  into  $n$  regular isotropic voxels, then

$$P_j(T(V)) = \sum_{i=1}^n \text{model}(p_j, o_j, x_i), \quad (8.2)$$

where  $\text{model}(p_j, o_j, x_i)$  denotes the model from [222], that is the contribution of voxel  $x_i$  to the measurement of the detector located at position/orientation  $(p_j, o_j)$ .

The system matrix  $A = [a_{ji}]$  is constructed with  $a_{ji} = \text{model}(p_j, o_j, x_i)$ . Each row corresponds to one measurement and each column corresponds to one voxel in the volume. The updated volume  $\hat{T}(V)$  can be computed from the following equation:

$$A\hat{T}(V) = c \quad (8.3)$$

The system matrix is used to simulate the moving image  $c'$  from the deformable volume  $T(V)$  that is initialized with the input volume  $V$  provided by pre- or intra-operative SPECT.

#### 8.2.5. Node Segmentation

The tissue around the radioactively tagged lymph nodes influences and possibly restricts the movement of the marked nodes, but the tissue is invisible on the SPECT images since it does not take up (enough) radioactivity. Lymph nodes are distributed as separate, spaced-apart entities embedded in that soft tissue throughout the body. Our deformation model thus only incorporates information that is readily available using the functional imaging modality SPECT.

Therefore, each lymph node is modeled as moving (for example due to tissue manipulation) independently from each other. Shape changes are not modeled since they cannot be detected by the gamma detector nor do they influence the localization of the lymph nodes due to their small size. Furthermore, only a limited area of the input volume is covered by the registration scan, possibly excluding some nodes. Due to the way the surgeons perform their search scan starting from the position indicated by the pre-operative SPECT, the scan covers the original and the new position of the lymph node in question. This also ensures that only one node is covered and by a sufficient number of measurements, avoiding ill-disposed registrations. Lymph nodes that are not covered by measurements are left as fixed and not moved during registration, whereas the lymph node that is covered by measurements are considered moving, for example because of tissue deformation.

The division of the set of nodes into a movable node and fixed nodes is automated for a smoother workflow. We use the contribution  $P_j$  of every voxel in the reconstruction volume towards the measurements (the information that is available for this voxel). The contribution/information for node  $N_i$  is then the following:

$$P_{N_i} = \sum_{k \in N_i} \sum_{j=1}^m \text{model}(p_j, o_j, x_k), \quad (8.4)$$

With this information, only the node with the most contribution is considered as movable, whereas all the other nodes are considered fixed (these thresholds have been determined experimentally). See figure 8.4 for an example using a phantom dataset and a short exploratory scan.



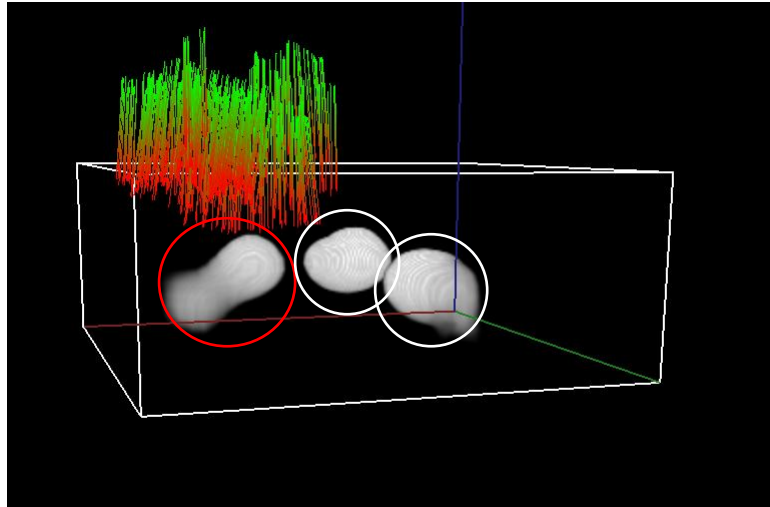


Figure 8.4.: This picture shows a visualization of three lymph nodes from a phantom experiment as well as the gamma detector measurements indicated by the red-green lines. The node enclosed by the red circle is added to the set of movable nodes. The two nodes enclosed by white circles on the other hand are left fixed.

The segmentation of the nodes is performed in an almost automatic manner. The only required input is the number of nodes that have to be detected. The segmentation is based on a thresholding algorithm combined with region growing [223]. It also serves as implicit regularization on the voxel level, since the nodes are moved as whole units. (Additional regularization in order to constrain the movement of the different nodes can be implemented, but has not been necessary according to our experiments.)

### 8.2.6. Decay Correction

Since the registration performs a simulation of the measurements from the deformed volume, the radioactive decay has to be considered according to the following equation:

$$K = K_0 e^{-\frac{t}{t_{\frac{1}{2}}}} \quad (8.5)$$

In this equation  $K_0$  is the number of the counts at the beginning of the time step,  $K$  the number of counts after the time step,  $t$  the time passed and  $t_{\frac{1}{2}}$  the half-life of the element, in the case of Technetium-99m 6 hours.

### 8.2.7. The Registration Pipeline for Intra-operative and Pre-operative SPECT

On the whole, the registration pipeline using fh-SPECT volumes as input  $V$  works in the following way: the radioactively labeled nodes are segmented, and the nodes as well as the set of measurements  $B$  are then used as input for the 1D-3D registration, resulting in the updated volume  $\hat{T}(V)$ . Using pre-operative SPECT scans works in principle just the same as using intra-operative SPECT scans as input, but requires a few more steps in order to register the pre-operative SPECT to the freehand coordinate system. Radio-opaque markers, visible in the co-registered CT, are used with a

standard rigid registration approach for the registration of the pre-operative SPECT and the freehand measurements.

### 8.3. Experiments

We have performed Monte–Carlo experiments, phantom experiments as well as an experiment on intra-operative data in order to evaluate our method.

#### 8.3.1. Monte–Carlo Experiments

The digital phantom for the Monte–Carlo experiments is a model of the physical phantom used later (see section 8.3.2). The phantom consists of three hollow spheres that are filled with a radioactive marker as can be seen in figure 8.5. One of the spheres is then moved by 25 mm (ground error) in order to simulate tissue deformation.

In this series of Monte–Carlo experiments, the limits of the accuracy of our approach are evaluated. The simulation of the count rates associated with the input measurements is performed by the open source Monte Carlo simulation GATE [224], developed by the OpenGATE Collaboration, and therefore independent of our detector "model".

The GATE software receives  $s$  nodes  $N_0, \dots, N_s$  as input with their associated radioactivity  $R_0, \dots, R_s$  as well as the set of detector positions  $\{(p_j, o_j) | 0 \leq j < s\}$ . The output is the set of simulated count rates  $\{c_j | 0 \leq j < s\}$ .

The Monte–Carlo experiments allow assessing the algorithms accurately without handling actual radioactivity. Furthermore, simulations allow excluding certain classes of errors that appear in the actual application like tracking errors or misalignment of the SPECT volume. This allows a first assessment of the theoretical limits of the algorithms.

#### Accuracy of Freehand Scanning Patterns

The first set of experiments (A-1) evaluates the accuracy that is achievable with realistic freehand scanning patterns. We captured 8 real freehand scanning patterns from experienced operators, as they are performing exploratory scans of the physical counterpart of our simulated box phantom. For each of these 8 recorded patterns, four Monte-Carlo simulations are run, resulting in 32 experiments.

Each of the search scans contains about 500 measurements resulting from a 30 second scanning time. Each measurement covers a duration of 0.06 seconds consistent with the real-world scans.

The region of interest is discretized as a 50 x 50 x 25 volume with an isotropic voxel size of 2.5 mm.

The input to the registration framework consists of a synthetically created volume of nuclear activity in the ROI before the deformation, the number of nodes in the input volume and the measurements of the volume after deformation that have been generated by OpenGATE.

This experiment is used in order to evaluate the registration performance under ideal conditions. There is no tracking error and the error introduced is only influenced by the probe model, the chosen registration algorithm (similarity measure and optimization procedure), as well as the inherent stochastic nature of the radioactive decay process given the limited amount of samples.

#### Positional Error

The next set of experiments examines the influence of positional errors, compared to an ideal situation where these errors do not exist. The first error we consider (A-2) is inaccurate prior knowledge, for example a registration error of the freehand SPECT volume. We simulate this by translating all nodes by 5 mm away from their original position. We also simulate tracking errors (A-3, A-4) by perturbing the measurement positions by a random distance of up 1.5 mm (A-3) or up to 3mm (A-4).

label	measurements	pattern	comment
A-1	simulated	recorded	
A-2	simulated	recorded	perturbed input volume (5 mm)
A-3	simulated	recorded	perturbed detector positions (1.5 mm)
A-4	simulated	recorded	perturbed detector positions (3 mm)
A-5	simulated	synthetic	
B-1	recorded	recorded	one-sided pattern
B-2	recorded	recorded	two-sided pattern
B-3	recorded	recorded	B-1 and B-2
B-4	recorded	recorded	full scan

Table 8.1.: This table gives an overview of the different experiments that have been performed.

### Distance dependency

Another important question is whether the registration accuracy decreases with an increasing distance between the node in the input volume and the position of the node in the volume under examination. For this experiment (A-5), synthetic scanning patterns of 500 measurements are used to cover the area of the original position up to the final position of the node. The final position is 1 cm, 2 cm or 3 cm away from the original position.

### 8.3.2. Phantom Experiments

The following sets of experiments use a phantom setup with radioactivity for more realistic experiments. The phantom consists of a plastic cubic shape, which also mounts the tracking target with reflective markers. The rectangular base plate is covered with a regular grid of bore holes that are spaced exactly 5 mm apart. Three hollow spheres with a radius of 9.86 mm each are mounted at mid-height in a triangular fashion with a separation of 4 cm each on the base plate. A Technetium-99m solution is filled into these spheres to simulate radioactive lymph nodes. The three spheres contained an activity of 1.6 MBq, 1.6 MBq and 1.65 MBq.

No attenuating medium has been used. One of the spheres could be moved away from the other two spheres by 2 cm (ground error) to simulate deformation during surgery, see figure 8.5 for an illustration. SPECT/CT imaging was provided by a Symbia T6 system (Siemens Healthcare, Germany), intra-operative SPECT by the declipseSPECT cart system (SurgicEye, Germany). The gamma detector employed was a gamma-detector System (Crystal Photonics, Germany), and tracking was performed using a Polaris Vicra infrared optical tracking system (Northern Digital, Canada) with reflective markers attached to both the gamma detector and to the scanning target (phantom or patient).

30 short scans of 30 seconds with about 500 measurements are performed, one set of 18 scans (experiment B-1) covering one side of the phantom, and another set of 12 scans (experiment B-2) covering two orthogonal sides of the phantom. Experiment B-3 combines the measurements from B-1 and B-2. The shorter scans from B-1 and B-2 cover the original position of the moved sphere as well as the current position.

Additionally, 23 full scans (experiment B-4) have been performed by two operators, covering the phantom from three sides in order to compare a full tomographic reconstruction to the 1D-3D registration. For each scan 3000 measurements are recorded, 1000 per each side, resulting in a scanning time of 3 minutes per scan. The full scans are reconstructed using Maximum Likelihood Expectation Maximization (MLEM) as the reconstruction algorithm with 20 iterations. A reconstructed volume from a full scan is used as the input volume for the registration of the shorter scans after the volume has been convolved with a standard Gaussian filter with  $\sigma = 3\text{mm}$ . The volume contains  $30 \times 30 \times 25$

voxels with an isotropic voxel size of 2.5 mm.

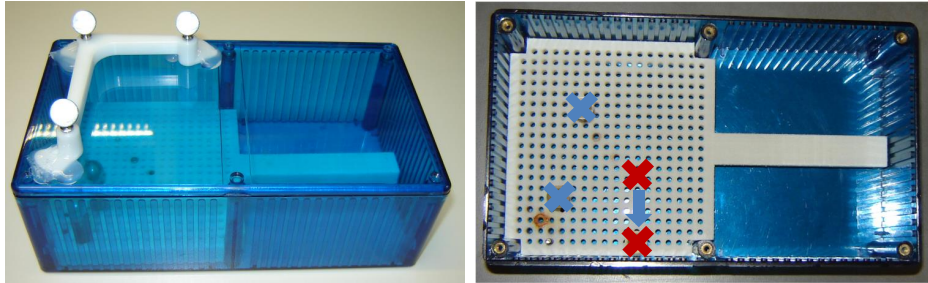


Figure 8.5.: The left shows the phantom used for the experiments. On the right the positions of the spheres with the radioactive marker are overlaid on the phantom. The red crosses mark the positions of the sphere that has been moved in order to simulate deformation.

### Minimum Number of Measurements

The first set of phantom experiments tackles the question of how many measurements are necessary for an accurate registration result. For this question, we use the measurements from the experiment B-3 and randomly select fewer measurements from 500 to 50 from each scan for the registration, resulting in 32 registrations for each of the different numbers of measurements.

### Scanning Patterns

Another potential source of error in the 1D-3D registration are the scanning patterns. We use the different scanning patterns from the experiments B-1 and B-2 in order to evaluate the influence of the pattern on the registration.

### Comparison to Reconstruction

The final question is how the registration approach compares in accuracy to a full reconstruction, as this approach is already used in the operating room. In order to compare the two approaches, the 12 full scans from experiment B-4 (approximately 3000 measurements, scanned from 3 sides) are reconstructed using the same model for the gamma detector that is used for the registration with only 500 measurements from experiment B-3.

### 8.3.3. Intra-operative Data

This section serves to demonstrate the benefits of registration in an actual clinical case. The measurements for the demonstration were recorded intra-operatively during a sentinel lymph node biopsy for the treatment of cervical cancer. The available data includes a 2D scintigraphy, a pre-operative SPECT/CT scan as well as intra-operative measurements from the gamma detector. Figure 8.6 shows the scintigraphy with two radioactively marked sentinel lymph nodes on the right and the left side of the patient.

The patient is injected with a radioactive marker (99m Technetium) and a scintigraphy as well as a SPECT/CT scan is performed. Before the SPECT/CT scan is performed, 6 CT markers with high radio-opacity are attached on the patient which later will be used for point based registration between

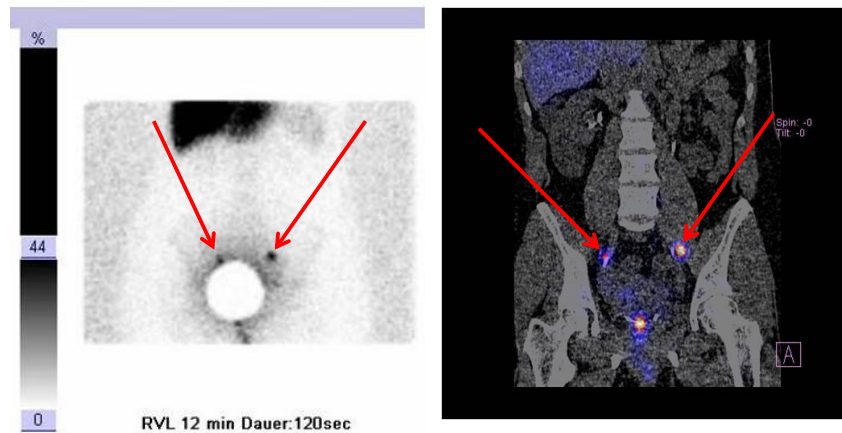


Figure 8.6.: The left side shows the scintigraphy with two lymph nodes visible as they are marked by the red arrows. The right shows a fusion of the SPECT and CT data showing the two lymph nodes as well as the injection site near the tumor.

the SPECT/CT data and the surgical site. The registration between the SPECT and the CT scan is provided by the hybrid scanner.

After opening the patient, the sentinel lymph node on the left side is located first. Since there is an offset of approximately 2 cm between the location of the lymph node as indicated by the pre-incision registration and its actual location, a fh-SPECT reconstruction is performed. Then the sentinel lymph node is removed. The right sentinel lymph node is likewise located more precisely by another fh-SPECT reconstruction.

For the following experiments the measurements of the first fh-SPECT scan before the resection as well as the SPECT volume are used as inputs for the 1D-3D registration. The time elapsed between the SPECT/CT scan and the freehand scan during surgery is 18 hours (half-life of Tc99m is 6 hours, resulting in a decay factor of 8).

## 8.4. Results

The evaluation of the experiments described above uses two different error measures, the 3D error and 2D error. The 3D error  $e_{3D} = |x_g - x_r|$  is the euclidean distance between  $x_g$ , the ground truth position of the movable node, and  $x_r$ , the position of the same node according to the registration. The 2D error is the error restricted to a plane orthogonal to the surgeon's view (see figure 8.8 for an illustration) determined by the average direction of the measurements. The surgeon is mostly interested in this 2D error, since the error in this plane determines the position of the cut, whereas the 3D error mainly determines how deep the cut has to be.

For illustrative purposes, we also perform an exhaustive computation of the cost function used in the registration algorithm. Given one movable node, the movable node is shifted to every voxel position and the resulting cost evaluated. Planes through the resulting 3D cost functions for different setups are shown in figure 8.9.

### 8.4.1. Monte-Carlo Experiments

The Monte-Carlo experiments are evaluated against the ground truth that has been used to perform the simulations.

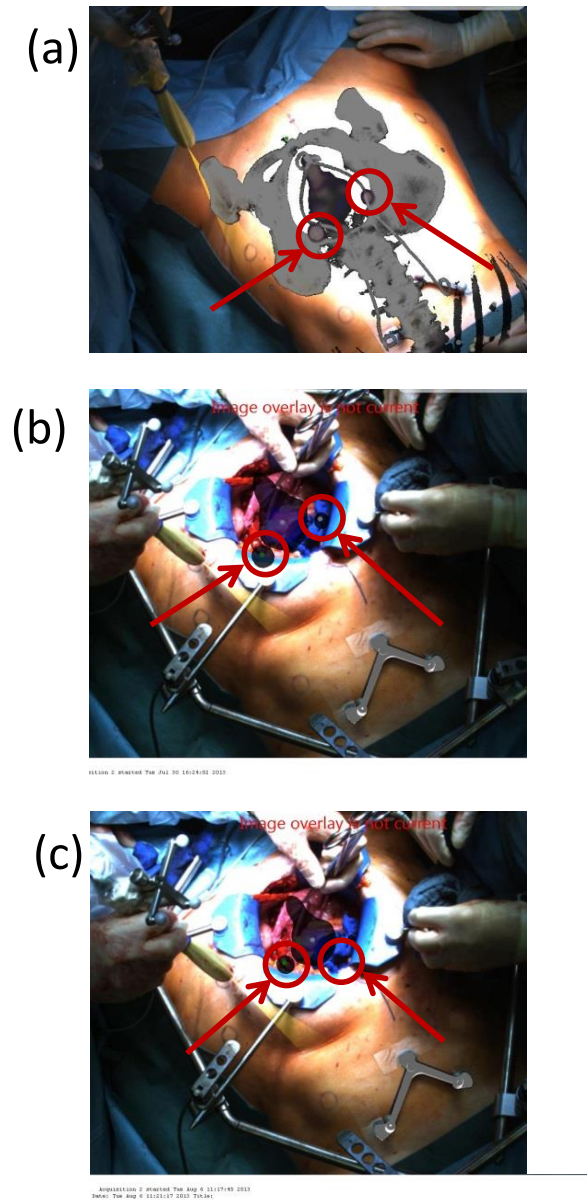


Figure 8.7.: The image (a) shows an overlay of the pre-operative SPECT and CT used as guidance during surgery, (b) shows SPECT guidance alone as it would appear during surgery, (c) shows the corrected position after 1D-3D registration. The red circles with the arrows indicate the positions of the lymph nodes.

### Accuracy of Freehand Scanning Patterns

Experiment A-1 evaluates the performance under ideal conditions. With a ground error of 25 mm, the average 2D error is 2.3 mm (with a standard deviation of 1.4 mm), and the average 3D error is 8.3 mm (with a standard deviation of 2.4mm), see also figure 8.10.

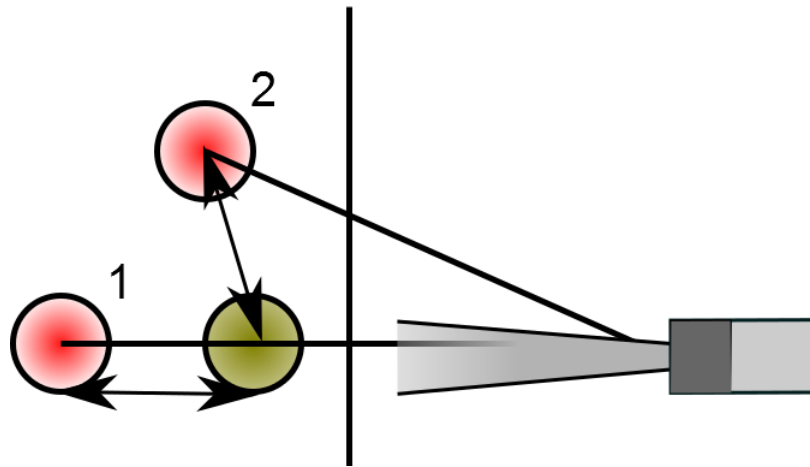


Figure 8.8.: This diagram illustrates the reasoning behind using the 2D and 3D errors: the nodes marked in red both have the same distance to the green node. However, the error in case 1 influences the depth but not the direction of the cut.

### Positional Errors

Figure 8.10 gives an overview of the results of the experiments A-2 to A-4 with 2D errors ranging from 2mm to 6mm (with a standard deviation between 1mm and 1.7mm) and 3D errors between 6mm and 8mm (with a standard deviation between 1.8mm and 2.1mm).

### Distance Dependency

The resulting error for the distance dependency experiment is ca. 2 mm for the 2D error and ca. 3 mm for the 3D error without any significant differences between the various distances, and a standard deviation of up to 1mm.

## 8.4.2. Phantom Experiments

For the evaluation of the phantom experiments a co-registered set of CT and SPECT scans is used.

### Minimum Number of Measurements

Figure 8.11 shows the results of the experiment using a decreasing number of measurements. We show the percentage of registrations that reach an accuracy of 10 mm or better, since this is the precision that is necessary for interventional use.

### Scanning Patterns

The result of the registration according to different scanning patterns can be seen in figure 8.12. The 2D error ranges from 4.2 mm to 4.5 mm with a standard deviation between 1.4mm and 2.3mm. The 3D error ranges from 5.6mm to 7.7mm with a standard deviation between 2.7mm and 4.8mm.

### Comparison to Reconstruction

The errors are once again computed in relation to the ground truth provided by the CT-scan. As can be seen in figure 8.13, the 2D error is larger for the registration than for the reconstruction with 4.4mm

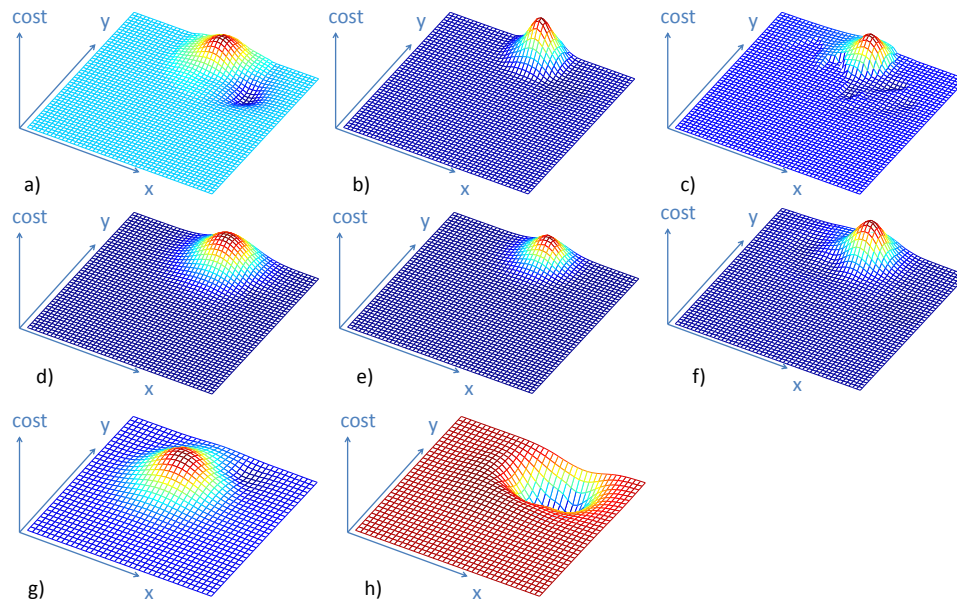


Figure 8.9.: The maxima in the plots represent optima of the cost function. The first row (a-c) shows cost functions resulting from the GATE simulation, the row (d-f) below shows the same scanning patterns, this time using the real measurement values from the phantom experiments. The last row (g and h) finally displays the cost function for the clinical case. Figure g shows the correct cost function, whereas the figure h shows the cost function without decay correction.

(and a standard deviation of 2mm) compared to 2mm (with a standard deviation of 1.6mm), while the 3D error is smaller, with 6.4mm (and a standard deviation of 3.5mm) compared to 7.1mm (with a standard deviation of 5mm), but the error stays below 10mm in both cases.

### 8.4.3. Intra-operative Data

Figure 8.7 shows the intra-operative guidance provided during the intervention with fused SPECT and CT data. We also recreated the overlay with the original uncorrected SPECT data (in the middle) and the updated SPECT data (on the right).

Due to the nature of intra-operative data, a ground truth for comparison is not available. However, the heat map in figure 8.14 shows a good correspondence between the registration result and the measurements of the highest amount of radioactivity. The updated position of the lymph node for intra-operative guidance in figure 8.7 also shows good correspondence with the description of the changed position the surgeons provided during surgery.



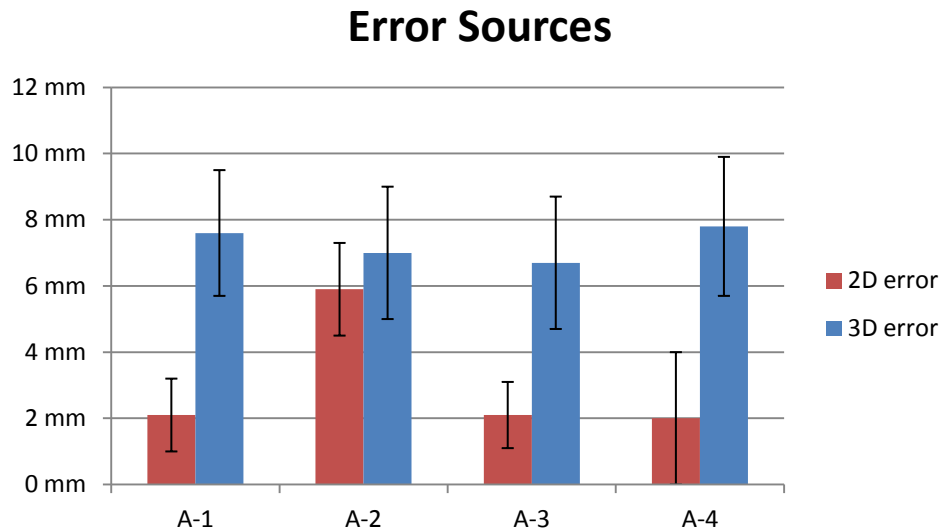


Figure 8.10.: This graph summarizes the results of the Monte–Carlo experiments (A-1 to A-4). Each row is the result of 32 registration operations. The ground error is 25 mm.

## 8.5. Discussion

### 8.5.1. Monte-Carlo Experiments

#### Positional Errors

The algorithm performs very well under the simulated ideal conditions, with an average 2D error of just 2.3mm. The accuracy is not only limited by the algorithm and the stochastic nature of radio-active decay, but also by the resolution of the input volume that is used as prior knowledge. This volume has a voxel size of 2.5 mm and presents a lower limit to the accuracy that can be achieved. Another factor that limits the accuracy is the small viewing angle due to the measurements that are almost parallel. This small viewing angle explains the larger 3D error (see also section 8.5.2). Even with artificially introduced errors, the resulting error remains below 10mm.

#### Distance dependency

This experiment shows that different realistic distances are not a major concern for the registration, as long as the original position as well as the new position of the node are covered by a sufficient amount of measurements.

### 8.5.2. Phantom Experiments

#### Minimum Number of Measurements

In order to be acceptable for surgical intervention, the error should stay below 10 mm. This is accomplished with 500 measurements in the phantom experiments. But even with 50 measurements, 80% of the registrations are accurate enough for the clinical application. Fewer measurements are an option depending on the time constraints during the surgical procedure. We would expect, however, that 500 measurements are acquired in most cases for more accurate results, considering that a scan

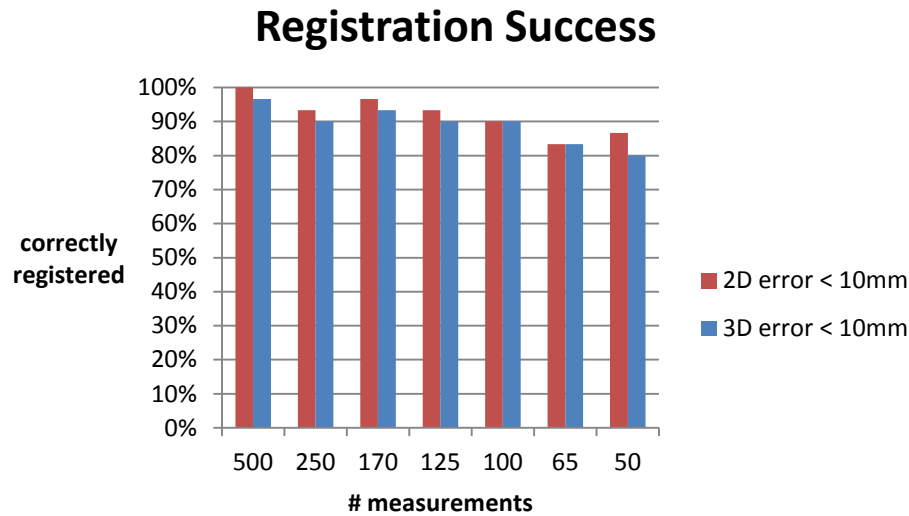


Figure 8.11.: This figure displays the number of adequate registrations (defined as less than 10 mm distance from ground truth). The ground error is 20mm.

duration of 30s is not exhaustive and switching between surgical instruments and the gamma detector takes some time as well.

### Scanning Patterns

As can be seen in figure 8.12 the difference between the different patterns is small for the 2D error, but noticeable for the 3D error. Due to the small viewing angle in experiment B-2, the depth information cannot be reliably deduced. However, the error remains below 10 mm and is therefore suitable for interventional use.

### Comparison to Reconstruction

The phantom experiments show that the registration approach is competitive to a pure reconstruction approach. The reconstruction achieves slightly better accuracy when considering the planar 2D errors, but at the cost of more measurements and therefore more time.

### 8.5.3. Intra–operative Data

Several sources of error affect the accuracy of the result: The rigid registration between the gamma detector position and the positions of the marker in the CT scan might not be accurate because of deformations of the soft tissue of the patient. Other sources of error include the optical tracking of the gamma detector relative to the patient, resolution limitations inherent in nuclear imaging, inaccuracies introduced by the 1D-3D registration, errors introduced by a simplified modeling of the physical properties of the gamma detector, as well as a bias introduced by the highly irregular manual scanning pattern.

While no ground truth is available that allows us to perform a strict quantification of the error, the good match of the registration to the observations of the surgeon as well as with the heat map indicates that the registration performs well in this case.

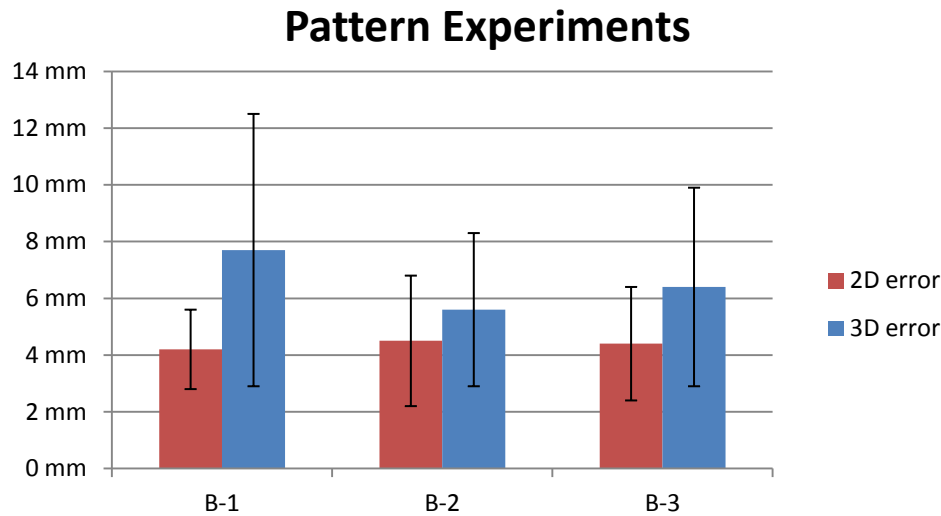


Figure 8.12.: The influence of the scanning pattern on the accuracy of the registration result is investigated in this set of experiments. The phantom scans are evaluated by comparing the registered results to the ground truth provided by a CT-scan. Two different scanning patterns are used: in one case, only one side of the phantom is scanned (B-1) or two sides are scanned (B-2). B-3 combined the results from B-1 and B-2. The ground error is 20mm.

Since the registration error in all the experiments is lower than 10 mm, the registration approach seems suitable for sentinel lymph node biopsies, even with artificially introduced errors in the input volume or the measurement data. Using a scanning pattern that covers two sides of the region of interest also leads to favorable registration results.

#### 8.5.4. Future Directions

Combining this research with robotically guided imaging is an interesting avenue of research, as more precise geometric positioning as well as a more regular sampling pattern lead to more accurate and more consistent reconstruction results [225]. We would expect that this transfers to our registration approach as well. Another direction for optimizing the registration result is a tailored user guidance that does not only take the coverage itself into account, but also whether voxels in the region of interest are covered from different directions. The scanning patterns can also be adapted to the measurements as they are recorded, as has been done for reconstruction [226].

The development of handheld 2D gamma cameras would result in a larger coverage of the area of interest as well as better count statistics in the same scanning time resulting in more accurate results as well [227].

Another exciting development are sensors integrated directly into surgical instruments. This approach allows a continuous real-time update for the visual image guidance during the surgical intervention without the need to switch between surgical instruments and detectors. This might also benefit from a continuous update of the registration as more and more measurements become available. Lowering the number of measurements as discussed in section 8.4.2 would prove to be very helpful in this use case.

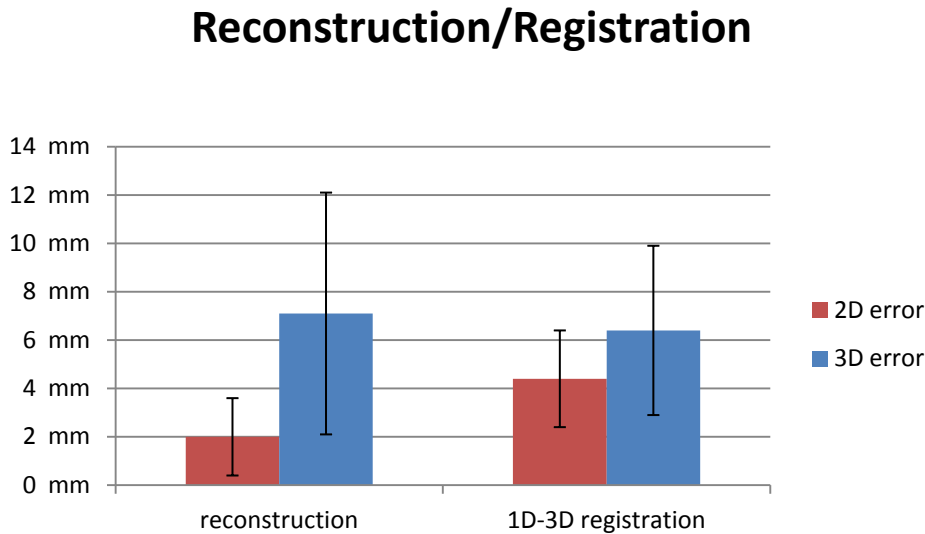


Figure 8.13.: For comparison to using reconstruction, full scans (3000 measurements) have been used to reconstruct the location of the node for the phantom experiments. The registration results used for comparison are the results from experiment B-3 from figure 8.12. The ground error is 20mm.

## 8.6. Conclusion

This chapter has described a novel 1D-3D registration procedure that registers 1D intra-operative measurements to a pre- or intra-operative 3D volume. We have kept the experiments clinically relevant with a scan time of less than 30s, resulting in less than 500 measurements per scan. The scan geometries have been kept to one-sided or two-sided scans at most, as can be reasonably expected during an intervention. For the Monte-Carlo simulations, previously acquired exploratory scans have been replayed. The accuracy of 5 mm - 10 mm is suitable for radio-guided surgery. In contrast to a reconstruction that requires a full scan, our registration-based method requires much fewer measurements and is easier to integrate into an already tight surgical workflow.

This method can be used instead of fh-SPECT reconstruction. The shorter scan times as required by the registration approach allow more frequent updates of the image guidance. Additionally, updates using traditional SPECT volumes add additional information which allows updating image guidance even when the fh-SPECT reconstruction fails to reconstruct a suitable image.

In summary, we have provided a proof of concept for a novel method for intra-operative guidance during surgery on radioactively labeled tissue. The system requires input data (SPECT scan) and the same gamma detector that is already used for radio-location. The additional hardware consists of the optical tracking device and a PC, easily fitting into an operating room. The short scan times do not impede the tightly orchestrated surgical workflow, thus enabling a fast and effective solution for intra-operative guidance.

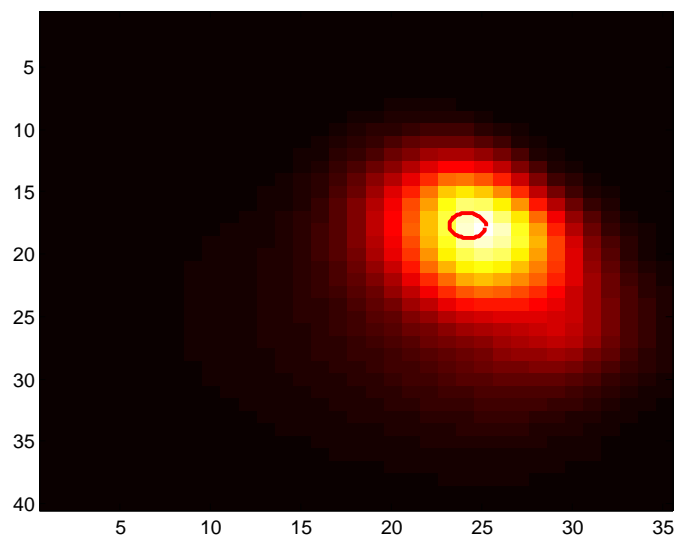


Figure 8.14.: The plot shows a heat map for the radioactivity. For each element of the plane the radioactivity is summed up. The circle indicates the location of the lymph node after the registration procedure and shows a good correspondence to the measured radioactivity.



**Part IV.**  
**Appendix**





# List of Figures

1.1. Dissertation Structure . . . . .	4
2.1. Stitching Histology Images . . . . .	10
2.2. Registration Overview . . . . .	11
2.3. Interpolation Grid . . . . .	12
2.4. Trilinear Interpolation . . . . .	12
2.5. Interpolation Functions . . . . .	13
2.6. Transformation Types . . . . .	13
2.7. Registration of Two CT Slices . . . . .	16
2.8. Error Metric for Offset Slices . . . . .	16
2.9. Joint Histogram for Mutual Information . . . . .	19
2.10. Comparison of Similarity Measures: Example Images . . . . .	20
2.11. Comparison of Similarity Measures: Graph . . . . .	20
2.12. Optimization Procedures . . . . .	21
2.13. Regularization for Medical Registration . . . . .	24
2.14. Regularization Strategies . . . . .	25
3.1. GeForce 6 Architecture . . . . .	30
3.2. OpenGL 1.0 Pipeline . . . . .	31
3.3. OpenGL 4.0 Pipeline . . . . .	32
3.4. Data Travelling through the Graphics Pipeline . . . . .	33
3.5. CPU-GPU Comparison . . . . .	34
3.6. Shader Programming Model . . . . .	35
3.7. Stream Programming Model . . . . .	35
3.8. Evolution of GPU Programming Languages . . . . .	36
3.9. The CUDA Programming Model . . . . .	37
3.10. CUDA GPU Architecture . . . . .	39
3.11. Cuda Thread Organization . . . . .	40
5.1. Registration Pipeline - Single Resolution . . . . .	52
5.2. GPU Registration Pipeline . . . . .	53
5.3. Learning-based Registration Prototype . . . . .	54
5.4. Vertex Texture Fetch Histograms . . . . .	55
5.5. VBO Histograms . . . . .	56
5.6. Performance . . . . .	57
5.7. Performance Improvement . . . . .	58
5.8. Registration Result . . . . .	59
5.9. Registration Screenshots . . . . .	59
5.10. Registration Screenshots 2 . . . . .	60
6.1. Preprocessing Step for Optimized GPU Histogram Computation . . . . .	65
6.2. Histogram Kernel . . . . .	66
6.3. Workload Distribution for Optimized GPU Histograms . . . . .	67
6.4. Partial Histograms . . . . .	68
6.5. Intensity Distribution and the Resulting Counter Distribution . . . . .	69

## List of Figures

---

6.6. Timing Experiments . . . . .	70
6.7. Timing: Time versus Volume Size . . . . .	71
6.8. Timing: Time versus Bin Size . . . . .	71
6.9. Example Data for Registration Timing . . . . .	72
6.10. Registration Result . . . . .	72
7.1. Transmission Emission Tomography . . . . .	75
7.2. SPECT Scanner Setup . . . . .	76
7.3. Maximum Likelihood Expectation Maximization (MLEM) . . . . .	79
7.4. Point Spread Function in SPECT Imaging . . . . .	81
7.5. GPU-Based Registration: Inner Loop . . . . .	82
7.6. Attenuation Map . . . . .	83
7.7. Forward Projection . . . . .	83
7.8. SPECT Projection on the GPU . . . . .	84
8.1. intra-operative freehand scan . . . . .	88
8.2. Sentinel Lymph Node Biopsy (SLNB) . . . . .	90
8.3. The Surgical Workflow . . . . .	91
8.4. Dividing Nodes into Movable and Fixed Nodes . . . . .	93
8.5. The phantom setup . . . . .	96
8.6. Clinical Case: Scintigraphy and SPECT/CT fusion . . . . .	97
8.7. SPECT/CT Overlay . . . . .	98
8.8. 2D error versus 3D error . . . . .	99
8.9. Cost functions for 1D-3D Registration . . . . .	100
8.10. Results of the Monte-Carlo experiments . . . . .	101
8.11. Error vs Number of Measurements . . . . .	102
8.12. Influence of the Scanning Pattern on the Registration Accuracy . . . . .	103
8.13. 1D-3D registration versus reconstruction . . . . .	104
8.14. Heat Map for Clinical Case . . . . .	105

# List of Tables

2.1. Reported Implementations of Common Similarity Measures and their Characteristics . . .	21
2.2. Abbreviations for Similarity Measures and References . . . . .	22
3.1. CUDA vs OpenCL Terms . . . . .	38
4.1. GPU Implementations of Common Similarity Measures . . . . .	46
8.1. Experiment Overview . . . . .	95



## A. Acronyms

API	Application programming interface
AR	Augmented reality
ART	Algebraic reconstruction Technique
CAS	Computer Aided Surgery
CC	Correlation Coefficient
CUDA	Compute Unified Device Architecture
CPU	Central Processing Unit
CR	Correlation Ratio
CT	Computed tomography
DOF	Degrees of freedom
DRR	Digitally reconstructed radiograph
DSA	Digital subtraction angiography
DVR	direct volume rendering
FBP	Filtered backprojection
FFD	Free form deformation
FLE	Fiducial localization error
FPGA	Field-programmable gate array
FRE	Fiducial registration error
FOV	Field of view
GC	gradient correlation
GD	gradient difference
GDGP	gradient proximity
GLSL	OpenGL Shading Language
GPU	Graphics processing unit
GPGPU	General Purpose GPU
GUI	Graphical user interface
HLSL	High Level Shading Language
IGI	Image-guided intervention
IGS	Image-guided surgery
JG	joint gradient
KL	Kullback-Leibler
LNC	local normalized correlation
MI	Mutual information
MRI	Magnetic resonance imaging
MLEM	Maximum Likelihood Expectation maximization
MPR	multiplanar reformatting
MRI	Magnetic resonance imaging
NMI	Normalized mutual information
NCC	Normalized Cross Correlation
NGF	Normalized Gradient Fields
NURBS	Non-uniform rational B-splines
OpenCL	Open Computing Language
OpenGL	Open Graphics Library

## A. Acronyms

---

OR	Operating room
PDE	Partial differential equation
PDF	probability density functions
PET	Positron emission tomography
PI	pattern intensity
PSF	Point Spread Function
RIU	ratio of image uniformity
ROI	Region of Interest
SAD	Sum of absolute differences
SART	simultaneous algebraic reconstruction technique
SLNC	sum of local normalized correlation
SNLB	Sentinel Lymph Node Biopsy
SPECT	Single photon emission computed tomography
SPMD	Single Program Multiple Data
SSD	Sum of squared differences
VLSI	very large scale integration
VOD	variance of differences
VOI	Volume of Interest
VWC	variance-weighted sum of local normalized correlation

## B. List of Publications

- C. Vetter, C. Guetter, C. Xu, and R. Westermann, “Non-rigid multi-modal registration on the GPU,” Proceedings of SPIE 6512, pp. 651228-651228-8, 2007.
- Z. Fan, C. Vetter, C. Guetter, D. Yu, R. Westermann, C. Xu, and A. Kaufman, “Optimized GPU implementation of learning-based non-rigid multi-modal registration,” Proceedings of SPIE 6914, p. 69142Y, 2008.
- C. Vetter and R. Westermann, “SPECT reconstruction on the GPU,” Proceedings of SPIE 6913, pp. 69132R-69132R-9, 2008.
- C. Vetter, A. Kamen, P. Khurd, and R. Westermann, “A learning-based approach to evaluate registration success,” in Medical Imaging and Augmented Reality - 5th International Workshop, MIAR 2010, Beijing, China, September 19-20, 2010. Proceedings, H. Liao, E. Edwards, X. Pan, Y. Fan, and G.-Z. Yang, eds., Lecture Notes in Computer Science 6326, pp. 429-437, Springer, 2010.
- C. Vetter and R. Westermann, “Optimized GPU histograms for multi-modal registration,” in Proceedings of the 8th IEEE International Symposium on Biomedical Imaging: From Nano to Macro, ISBI 2011, March 30 - April 2, 2011, Chicago, Illinois, USA, pp. 1227-1230, IEEE, 2011.
- O. Fluck, C. Vetter, W. Wein, A. Kamen, B. Preim, and R. Westermann, “A survey of medical image registration on graphics hardware,” Computer Methods and Programs in Biomedicine 104(3), pp. e45 - e57, 2011.
- C. Vetter, T. Lasser, T. Wendler, and N. Navab, “1D-3D registration for functional nuclear imaging,” in Medical Image Computing and Computer-Assisted Intervention - MICCAI 2011, G. Fichtinger, A. Martel, and T. Peters, eds., Lecture Notes in Computer Science 6891, pp. 227-234, Springer, 2011.
- R. Toth, J. Chappelow, C. Vetter, O. Kutter, C. Russ, M. Feldman, J. Tomaszewski, N. Shih, and A. Madabhushi, “Incorporating the whole-mount prostate histology reconstruction program histostitcher into the extensible imaging platform (XIP) framework,” Proc. SPIE 8315, pp. 83151K-83151K-11, 2012.
- C. Vetter, T. Lasser, A. Okur, and N. Navab, “1D-3D registration for intra-operative nuclear imaging in radio-guided surgery,” Medical Imaging, IEEE Transactions on 34, pp. 608-617, Feb 2015.

*B. List of Publications*

---



## Bibliography

- [1] O. Fluck, C. Vetter, W. Wein, A. Kamen, B. Preim, and R. Westermann, "A survey of medical image registration on graphics hardware," *Computer Methods and Programs in Biomedicine* **104**(3), pp. e45 – e57, 2011.
- [2] C. Vetter, C. Guetter, C. Xu, and R. Westermann, "Non-rigid multi-modal registration on the GPU," *Proceedings of SPIE* **6512**, pp. 651228–651228–8, 2007.
- [3] Z. Fan, C. Vetter, C. Guetter, D. Yu, R. Westermann, A. Kaufman, and C. Xu, "Optimized GPU implementation of learning-based non-rigid multi-modal registration," *Proceedings of SPIE* **6914**, p. 69142Y, 2008.
- [4] C. Vetter and R. Westermann, "Optimized GPU histograms for multi-modal registration," in *Proceedings of the 8th IEEE International Symposium on Biomedical Imaging: From Nano to Macro, ISBI 2011, March 30 - April 2, 2011, Chicago, Illinois, USA*, pp. 1227–1230, IEEE, 2011.
- [5] C. Vetter and R. Westermann, "SPECT reconstruction on the GPU," *Proceedings of SPIE* **6913**, pp. 69132R–69132R–9, 2008.
- [6] C. Vetter, T. Lasser, T. Wendler, and N. Navab, "1D-3D registration for functional nuclear imaging," in *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2011*, G. Fichtinger, A. Martel, and T. Peters, eds., *Lecture Notes in Computer Science* **6891**, pp. 227–234, Springer, 2011.
- [7] C. Vetter, T. Lasser, A. Okur, and N. Navab, "1D-3D registration for intra-operative nuclear imaging in radio-guided surgery," *Medical Imaging, IEEE Transactions on* **34**, pp. 608–617, Feb 2015.
- [8] J. V. Hajnal, D. L. Hill, and D. J. Hawkes, *Medical Image registration*, CRC Press, 2001.
- [9] W. A. Kalender, *Computed Tomography: Fundamentals, System Technology, Image Quality, Applications*, Computed Tomography: Fundamentals, System Technology, Image Quality, Applications, by Willi A. Kalender, pp. 220. ISBN 3-89578-081-2. Wiley-VCH , November 2000., Nov. 2000.
- [10] R. B. Buxton, *Introduction to Functional Magnetic Resonance Imaging: Principles and Techniques*, Cambridge University Press, 2009.
- [11] D. Bailey, D. Townsend, P. Valk, and M.N.Maisey, *Positron Emission Tomography*, Springer, 2003.
- [12] M. Wernick and J. Aarsvold, *Emission Tomography: The Fundamentals of PET and SPECT*, Academic Press, 2004.
- [13] T. Peters and K. Cleary, eds., *Image-Guided Interventions: Technology and Applications*, Springer, 2008.
- [14] U. Pietrzyk, K. Herholz, A. Schuster, H.-M. v. Stockhausen, H. Lucht, and W.-D. Heiss, "Clinical applications of registration and fusion of multimodality brain images from PET, SPECT, CT, and MRI," *European Journal of Radiology* **21**(3), pp. 174 – 182, 1996.

- [15] R. Toth, J. Chappelow, C. Vetter, O. Kutter, C. Russ, M. Feldman, J. Tomaszewski, N. Shih, and A. Madabhushi, "Incorporating the whole-mount prostate histology reconstruction program histostitcher into the extensible imaging platform (XIP) framework," *Proc. SPIE* **8315**, pp. 83151K–83151K–11, 2012.
- [16] B. Fischer and J. Modersitzki, "Ill-posed medicine - an introduction to image registration," *Inverse Problems* **24**(3), p. 034008, 2008.
- [17] J. Maintz and M. A. Viergever, "A survey of medical image registration," *Medical Image Analysis* **2**(1), pp. 1 – 36, 1998.
- [18] B. Zitová and J. Flusser, "Image registration methods: a survey.," *Image Vision Comput.* **21**(11), pp. 977–1000, 2003.
- [19] J. Modersitzki, *Numerical Methods for Image Registration*, Oxford University Press, 2004.
- [20] A. A. Goshtasby, *Image Registration - Principles, Tools and Methods.*, Advances in Computer Vision and Pattern Recognition, Springer, 2012.
- [21] J. Tsao, "Interpolation artifacts in multimodality image registration based on maximization of mutual information," *IEEE Transactions on Medical Imaging* **22**, pp. 854–864, Jul 2003.
- [22] J. Pluim, "Interpolation Artefacts in Mutual Information-Based Image Registration," *Computer Vision and Image Understanding* **77**, pp. 211–232, Feb 2000.
- [23] G. Wolberg, *Digital Image Warping*, IEEE Computer Society Press, Los Alamitos, CA, USA, 1st ed., 1994.
- [24] T. Lehmann, C. Gonner, and K. Spitzer, "Survey: interpolation methods in medical image processing," *Medical Imaging, IEEE Transactions on* **18**, pp. 1049–1075, Nov 1999.
- [25] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics: Principles and Practice (2Nd Ed.)*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.
- [26] P. Markelj, D. Tomazevic, B. Likar, and F. Pernus, "A review of 3d/2d registration methods for image-guided interventions," *Medical Image Analysis* **16**(3), pp. 642 – 661, 2012. Computer Assisted Interventions.
- [27] S. Aouadi and L. Sarry, "Accurate and precise 2d-3d registration based on x-ray intensity," *Computer Vision and Image Understanding* **110**(1), pp. 134 – 151, 2008.
- [28] E. Bullitt, A. Liu, S. R. Aylward, C. Coffey, J. Stone, S. K. Mukherji, K. E. Muller, and S. M. Pizer, "Registration of 3d cerebral vessels with 2d digital angiograms: Clinical evaluation," *Academic Radiology* **6**(9), pp. 539 – 546, 1999.
- [29] A. Khamene, R. Chisu, W. Wein, N. Navab, and F. Sauer, "A novel projection based approach for medical image registration," in *Proceedings of the Third International Conference on Biomedical Image Registration, WBIR'06*, pp. 247–256, Springer, 2006.
- [30] S. Demirci, O. Kutter, F. Manstad-Hulaas, R. Bauernschmitt, and N. Navab, "Advanced 2D-3D registration for endovascular aortic interventions: Addressing dissimilarity in images," in *SPIE Medical Imaging*, (San Diego, California, USA), Feb 2008.
- [31] G. W. Sherouse, K. L. Novins, and E. L. Chaney, "Computation of digitally reconstructed radiographs for use in radiotherapy treatment design," *International Journal of Radiation Oncology, Biology and Physics* **18**(3), pp. 651–658, 1990.
- [32] T. W. Sederberg and S. R. Parry, "Free-form deformation of solid geometric models," in *Proceedings of the 13th annual conference on Computer graphics and interactive techniques, SIGGRAPH '86*, pp. 151–160, ACM, (New York, NY, USA), 1986.

- 
- [33] A. Roche, G. Malandain, and N. Ayache, “Unifying maximum likelihood approaches in medical image registration,” 1999.
- [34] A. Collignon, F. Maes, D. Delaere, P. Vandermeulen, P. Suetens, and G. Marchal, “Automated multimodality image registration based on information theory,” in *Proceedings of Information Processing in Medical Imaging*, Y. Bizais, C. Barillot, and R. Di Paola, eds., *Lecture Notes in Computer Science* **3**, pp. 263–274, Kluwer Academic Publishers, (Ile de Berder, France), June 1995.
- [35] W. M. Wells, P. A. Viola, H. Atsumi, S. Nakajima, and R. Kikinis, “Multi-modal volume registration by maximization of mutual information,” *Medical Image Analysis* **1**, pp. 35–51, 1996.
- [36] P. Viola and W. M. Wells, III, “Alignment by maximization of mutual information,” *Int. J. Comput. Vision* **24**, pp. 137–154, Sept. 1997.
- [37] F. Maes, A. Collignon, D. Vandermeulen, G. Marchal, and P. Suetens, “Multimodality image registration by maximization of mutual information,” *Medical Imaging, IEEE Transactions on* **16**, pp. 187–198, Apr 1997.
- [38] D. L. Hill, D. J. Hawkes, N. A. Harrison, and C. F. Ruff, “A strategy for automated multimodality image registration incorporating anatomical knowledge and image characteristics,” in *Information Processing in Medical Imaging*, H. H. Barrett and A. Gmitro, eds., *Lecture Notes in Computer Science* **687**, pp. 182–196, Springer Berlin Heidelberg, 1993.
- [39] P. Viola, *Alignment by Maximization of Mutual Information*. PhD thesis, Massachusetts Institute of Technology, June 1995.
- [40] C. Studholme, D. L. G. Hill, and D. J. Hawkes, “An overlap invariant entropy measure of 3D medical image alignment,” *Pattern Recognition* **32**(1), pp. 71–86, 1999.
- [41] G. Penney, J. Weese, J. Little, P. Desmedt, D. Hill, and D. Hawkes, “A comparison of similarity measures for use in 2-D-3-D medical image registration,” *IEEE Transactions on Medical Imaging* **17**, pp. 586–595, Aug 1998.
- [42] J. P. W. Pluim, J. Maintz, and M. Viergever, “Mutual-information-based registration of medical images: a survey,” *Medical Imaging, IEEE Transactions on* **22**, pp. 986–1004, Aug 2003.
- [43] A. C. S. Chung, W. M. Wells, III, A. Norbash, and W. E. L. Grimson, “Multi-modal image registration by minimising Kullback-Leibler distance,” in *Proceedings of the 5th International Conference on Medical Image Computing and Computer-Assisted Intervention-Part II, MICCAI '02*, pp. 525–532, Springer, (London, UK), 2002.
- [44] H. ming Chan, A. C. S. Chung, S. C. H. Yu, E. Norbash, and W. M. W. Iii, “Multi-modal image registration by minimizing Kullback-Leibler distance,” in *International Conference on Medical Image Computing and Computer Assisted Intervention*, pp. 525–532, Springer, 2002.
- [45] R. P. Woods, S. R. Cherry, and J. C. Mazziotta, “Rapid automated algorithm for aligning and reslicing PET images,” *J Comput Assist Tomogr* **16**(4), pp. 620–633, 1992.
- [46] J. Weese, T. M. Buzug, C. Lorenz, and C. Fassnacht, “An approach to 2D/3D registration of a vertebra in 2D X-ray fluoroscopies with 3D CT images,” in *CVRMed*, J. Troccaz, W. E. L. Grimson, and R. Mösges, eds., *Lecture Notes in Computer Science* **1205**, pp. 119–128, Springer, 1997.
- [47] A. Roche, G. Malandain, X. Pennec, and N. Ayache, “The correlation ratio as a new similarity measure for multimodal image registration,” in *Proceedings of the First International Conference on Medical Image Computing and Computer-Assisted Intervention, MICCAI '98*, pp. 1115–1124, Springer-Verlag, (London, UK), 1998.

- [48] A. Roche, G. Malandain, X. Pennec, and N. Ayache, “Multimodal image registration by maximization of the correlation ratio,” tech. rep., INRIA, 1998.
- [49] A. Roche, X. Pennec, G. Malandain, and N. Ayache, “Rigid registration of 3-d ultrasound with mr images: a new approach combining intensity and gradient information,” *Medical Imaging, IEEE Transactions on* **20**, pp. 1038–1049, Oct 2001.
- [50] D. Sarrut and S. Clippe, “Geometrical transformation approximation for 2D/3D intensity-based registration of portal images and CT scan,” in *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2001*, W. Niessen and M. Viergever, eds., *Lecture Notes in Computer Science* **2208**, pp. 532–540, Springer Berlin Heidelberg, 2001.
- [51] D. LaRose, *Iterative X-ray/CT Registration Using Accelerated Volume Rendering*. PhD thesis, Robotics Institute, Carnegie Mellon University, (Pittsburgh, PA), May 2001.
- [52] X. Mei and F. Porikli, “Fast image registration via joint gradient maximization: application to multimodal data,” *Proc. SPIE* **6395**, pp. 63950P–63950P–5, 2006.
- [53] J. H. Hipwell, G. P. Penney, T. C. Cox, J. V. Byrne, and D. J. Hawkes, “2D-3D intensity based registration of DSA and MRA - a comparison of similarity measures,” in *Medical Image Computing and Computer-Assisted Intervention — MICCAI 2002*, T. Dohi and R. Kikinis, eds., *Lecture Notes in Computer Science* **2489**, pp. 501–508, Springer Berlin Heidelberg, 2002.
- [54] M. Holden, D. L. G. Hill, E. R. E. Denton, J. M. Jarosz, T. C. S. Cox, T. Rohlfing, J. Goodey, and D. J. Hawkes, “Voxel similarity measures for 3-D serial MR brain image registration,” *IEEE Transactions on Medical Imaging* **19**, pp. 94–102, 2000.
- [55] R. McLaughlin, J. Hipwell, D. Hawkes, J. Noble, J. Byrne, and T. Cox, “A comparison of a similarity-based and a feature-based 2-d-3-d registration method for neurointerventional use,” *Medical Imaging, IEEE Transactions on* **24**, pp. 1058–1066, aug. 2005.
- [56] J. H. Hipwell, G. P. Penney, R. A. McLaughlin, K. S. Rhode, P. E. Summers, T. C. S. Cox, J. V. Byrne, J. A. Noble, and D. J. Hawkes, “Intensity based 2d-3d registration of cerebral angiograms,” *IEEE Trans. Med. Imaging* **22**(11), pp. 1417–1426, 2003.
- [57] A. Khamene, P. Bloch, W. Wein, M. Svatos, and F. Sauer, “Automatic registration of portal images and volumetric CT for patient positioning in radiation therapy,” *Medical Image Analysis* **10**, pp. 96–112, Feb. 2006.
- [58] G. S. Cox and G. de Jager, “Automatic registration of temporal image pairs for digital subtraction angiography,” *Proc. SPIE* **2167**, pp. 188–199, 1994.
- [59] A. Kubias, F. Deinzer, T. Feldmann, S. Paulus, D. Paulus, B. Schreiber, and T. Brunner, “2d/3d image registration on the GPU,” *International Journal of Pattern Recognition and Image Analysis* **18**(3), pp. 381–389, 2008.
- [60] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, Cambridge University Press, New York, NY, USA, 3 ed., 2007.
- [61] C. Studholme, D. L. G. Hill, and D. J. Hawkes, “Automated three-dimensional registration of magnetic resonance and positron emission tomography brain images by multiresolution optimization of voxel similarity measures,” *Medical Physics* **24**(1), pp. 25–35, 1997.
- [62] J. A. Nelder and R. Mead, “A Simplex Method for Function Minimization,” *The Computer Journal* **7**, pp. 308–313, Jan. 1965.
- [63] M. J. D. Powell, “An efficient method for finding the minimum of a function of several variables without calculating derivatives,” *The Computer Journal* **7**(2), pp. 155–162, 1964.
- [64] R. Brent, *Algorithms for minimization without derivatives*, Prentice-Hall, 1973.

- 
- [65] J. R. Shewchuk, “An introduction to the conjugate gradient method without the agonizing pain,” tech. rep., Carnegie Mellon University, Pittsburgh, PA, USA, 1994.
- [66] J.-P. Thirion, “Image matching as a diffusion process: an analogy with Maxwell’s demons,” *Medical Image Analysis* **2**(3), pp. 243–260, 1998.
- [67] R. Stefanescu, X. Pennec, and N. Ayache, “Grid powered nonlinear image registration with locally adaptive regularization,” *MICCAI 2003 Special Issue* **8**, pp. 325–342, 2004.
- [68] M. Staring, S. Klein, and J. P. Pluim, “A rigidity penalty term for nonrigid registration,” *Medical Physics* **34**, pp. 4098 – 4108, Nov 2007.
- [69] E. Haber and J. Modersitzki, “Numerical methods for volume preserving image registration,” in *Inverse Problems, Institute of Physics Publishing*, pp. 1621–1638, 2004.
- [70] D. Rueckert, L. I. Sonoda, C. Hayes, D. L. G. Hill, M. O. Leach, and D. J. Hawkes, “Nonrigid registration using free-form deformations: Application to breast MR images,” *IEEE Transactions on Medical Imaging* **18**, pp. 712–721, 1999.
- [71] D. Zikic, M. Baust, A. Kamen, and N. Navab, “Generalization of deformable registration in Riemannian Sobolev spaces,” in *Proc. International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2010.
- [72] J. Modersitzki, *FAIR: Flexible Algorithms for Image Registration*, SIAM, Philadelphia, 2009.
- [73] W. Jones, L. Byars, and M. Casey, “Positron emission tomographic images and expectation maximization: a VLSI architecture for multiple iterations per second,” *Nuclear Science, IEEE Transactions on* **35**, pp. 620 – 624, Feb 1988.
- [74] C. Castro-Pareja, J. Jagadeesh, and R. Shekhar, “Fair: a hardware architecture for real-time 3-d image registration,” *Information Technology in Biomedicine, IEEE Transactions on* **7**, pp. 426–434, Dec 2003.
- [75] C. R. Castro-Pareja, J. M. Jagadeesh, and R. Shekhar, “Fpga-based acceleration of mutual information calculation for real-time 3d image registration,” *Proc. SPIE* **5297**, pp. 212–219, 2004.
- [76] O. Dandekar and R. Shekhar, “Fpga-accelerated deformable image registration for improved target-delineation during ct-guided interventions,” *Biomedical Circuits and Systems, IEEE Transactions on* **1**, pp. 116 –127, june 2007.
- [77] D. Stsepankou, K. Kornmesser, J. Hesser, and R. Manner, “Fpga-acceleration of cone-beam reconstruction for the x-ray ct,” in *Field-Programmable Technology, 2004. Proceedings. 2004 IEEE International Conference on*, pp. 327 – 330, dec. 2004.
- [78] J. Deng, B. Yan, J. Li, and L. Li, “Parallel no-waiting pipelining accelerating CT image reconstruction based on FPGA,” in *2010 3rd International Conference on Biomedical Engineering and Informatics (BMEI)*, **1**, pp. 451–455, Oct 2010.
- [79] P. Dillinger, J. Vogelbruch, J. Leinen, S. Suslov, R. Patzak, H. Winkler, and K. Schwan, “FPGA-based real-time image segmentation for medical systems and data processing,” *IEEE Transactions on Nuclear Science* **53**, pp. 2097–2101, Aug 2006.
- [80] S. Che, J. Li, J. Sheaffer, K. Skadron, and J. Lach, “Accelerating compute-intensive applications with gpus and fpgas,” in *Symposium on Application Specific Processors*, pp. 101–107, Jun 2008.
- [81] <http://www.top500.org/list/2012/06/200>.
- [82] K. Akeley, “RealityEngine graphics,” in *Proceedings of the 20th annual conference on Computer graphics and interactive techniques, SIGGRAPH ’93*, pp. 109–116, ACM, (New York, NY, USA), 1993.

- [83] J. S. Montrym, D. R. Baum, D. L. Dignam, and C. J. Migdal, “InfiniteReality: a real-time graphics system,” in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques, SIGGRAPH '97*, pp. 293–302, ACM Press/Addison-Wesley Publishing Co., (New York, NY, USA), 1997.
- [84] D. Shreiner, *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Seventh Edition*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1993.
- [85] E. Lindholm, M. J. Kilgard, and H. Moreton, “A user-programmable vertex engine,” in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques, SIGGRAPH '01*, pp. 149–158, ACM, (New York, NY, USA), 2001.
- [86] M. Corporation, *Microsoft DirectX 9 Programmable Graphics Pipeline*, Microsoft Press, Redmond, WA, USA, 2003.
- [87] J. Montrym and H. Moreton, “The GeForce 6800,” *Micro, IEEE* **25**, pp. 41 – 51, march-april 2005.
- [88] W. R. Mark, R. S. Glanville, K. Akeley, and M. J. Kilgard, “Cg: a system for programming graphics hardware in a C-like language,” *ACM Trans. Graph.* **22**(3), pp. 896–907, 2003.
- [89] R. J. Rost, *OpenGL(R) Shading Language (2nd Edition)*, Addison-Wesley Professional, January 2006.
- [90] E. Kilgariff and R. Fernando, “The geforce 6 series GPU architecture,” in *GPU Gems 2*, M. Pharr, ed., ch. 30, pp. 521–545, Addison Wesley, Mar. 2005.
- [91] D. Blythe, “The Direct3D 10 system,” *ACM Trans. Graph.* **25**(3), pp. 724–734, 2006.
- [92] T. Schiwietz, T.-c. Chang, P. Speier, and R. Westermann, “MR image reconstruction using the GPU,” *Proc. SPIE* **6142**, pp. 61423T–61423T–12, 2006.
- [93] J. Owens, “Streaming architectures and technology trends,” in *GPU Gems 2*, M. Pharr and R. Fernando, eds., Addison-Wesley Professional, 2005.
- [94] I. Buck, T. Foley, D. Horn, J. SUGERMAN, K. Fatahalian, M. Houston, and P. Hanrahan, “Brook for GPUs: stream computing on graphics hardware,” in *ACM SIGGRAPH 2004 Papers, SIGGRAPH '04*, pp. 777–786, ACM, (New York, NY, USA), 2004.
- [95] A. Lefohn, J. M. Kniss, R. Strzodka, S. Sengupta, and J. D. Owens, “Glift: Generic, efficient, random-access GPU data structures,” *ACM Transactions on Graphics* **25**, pp. 60–99, Jan 2006.
- [96] M. D. McCool, Z. Qin, and T. S. Popa, “Shader metaprogramming,” in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, HWWS '02*, pp. 57–68, Eurographics Association, (Aire-la-Ville, Switzerland, Switzerland), 2002.
- [97] M. McCool, S. Du Toit, T. Popa, B. Chan, and K. Moule, “Shader algebra,” in *ACM SIGGRAPH 2004 Papers, SIGGRAPH '04*, pp. 787–795, ACM, (New York, NY, USA), 2004.
- [98] J. Krüger and R. Westermann, “Linear algebra operators for GPU implementation of numerical algorithms,” in *ACM SIGGRAPH 2003 Papers, SIGGRAPH '03*, pp. 908–916, ACM, (New York, NY, USA), 2003.
- [99] A. Greß and G. Zachmann, “GPU-ABiSort: Optimal parallel sorting on stream architectures,” in *Proc. 20th IEEE Int'l Parallel and Distributed Processing Symposium (IPDPS)*, (Rhodes Island, Greece), Apr 2006.
- [100] N. K. Govindaraju, B. Lloyd, W. Wang, M. Lin, and D. Manocha, “Fast computation of database operations using graphics processors,” in *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pp. 215–226, ACM Press, (New York, NY, USA), 2004.

- 
- [101] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell, “A survey of general-purpose computation on graphics hardware,” in *Eurographics 2005, State of the Art Reports*, pp. 21–51, Aug 2005.
- [102] M. Peercy, M. Segal, and D. Gerstmann, “A performance-oriented data parallel virtual machine for GPUs,” in *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches*, p. 184, ACM Press, (New York, NY, USA), 2006.
- [103] ATI, “ATI stream computing - compute abstraction layer (CAL),” tech. rep., AMD, 2010.
- [104] T. Blank and J. Nickolls, “A Grimm collection of MIMD fairy tales,” in *Fourth Symposium on the Frontiers of Massively Parallel Computation, 1992*, pp. 448–457, Oct 1992.
- [105] AMD, “AMD accelerated parallel processing,” tech. rep., AMD, 2012.
- [106] NVidia, “NVIDIA CUDA - NVIDIA CUDA C programming guide - version 4.2,” tech. rep., NVidia, 2012.
- [107] Khronos OpenCL Working Group, *The OpenCL Specification, version 1.1*, 2010.
- [108] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, “NVIDIA Tesla: A unified graphics and computing architecture,” *Micro, IEEE* **28**, pp. 39–55, Mar-Apr 2008.
- [109] NVidia, “NVIDIA CUDA C programming best practices guide,” tech. rep., NVidia, 2010.
- [110] N. K. Govindaraju, B. Lloyd, Y. Dotsenko, B. Smith, and J. Manferdelli, “High performance discrete fourier transforms on graphics processors,” in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing, SC '08*, pp. 2:1–2:12, IEEE Press, (Piscataway, NJ, USA), 2008.
- [111] X. Xue, A. Cheryauka, and D. Tubbs, “Acceleration of fluoro-CT reconstruction for a mobile C-arm on GPU and FPGA hardware: a simulation study,” in *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, M. J. Flynn and J. Hsieh, eds., *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series* **6142**, pp. 1494–1501, Mar 2006.
- [112] N. Neophytou, F. Xu, and K. Mueller, “Hardware acceleration vs. algorithmic acceleration: can GPU-based processing beat complexity optimization for CT?,” *Medical Imaging 2007: Physics of Medical Imaging* **65105F**, p. 9, 2007.
- [113] W. Xu, F. Xu, M. Jones, B. Keszthelyi, J. Sedat, D. Agard, and K. Mueller, “High-performance iterative electron tomography reconstruction with long-object compensation using graphics processing units (GPUs),” *Journal of Structural Biology* **171**(2), pp. 142–153, 2010.
- [114] M. Silberstein, A. Schuster, D. Geiger, A. Patney, and J. D. Owens, “Efficient computation of sum-products on GPUs through software-managed cache,” in *Proceedings of the 22nd annual international conference on Supercomputing, ICS '08*, pp. 309–318, ACM, (New York, NY, USA), 2008.
- [115] P. B. Noël, A. Walczak, K. R. Hoffmann, J. Xu, b. bf Corso, and S. Schafer, “Clinical Evaluation of GPU-Based Cone Beam Computed Tomography,” in *Proceedings of High-Performance Medical Image Computing and Computer-Aided Intervention (HP-MICCAI)*, 2008.
- [116] J. Tolke and M. Krafczyk, “Teraflop computing on a desktop pc with gpus for 3d cfd,” *Int. J. Comput. Fluid Dyn.* **22**, pp. 443–456, Aug 2008.
- [117] Z. Yang, Y. Zhu, and Y. Pu, “Parallel image processing based on cuda,” in *Computer Science and Software Engineering, 2008 International Conference on*, **3**, pp. 198–201, 2008.

- [118] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupati, P. Hammarlund, R. Singhal, and P. Dubey, “Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU,” in *Proceedings of the 37th annual international symposium on Computer architecture, ISCA '10*, pp. 451–460, ACM, (New York, NY, USA), 2010.
- [119] NVidia, “NVIDIA’s next generation CUDA compute architecture: Fermi,” tech. rep., NVidia, 2009.
- [120] NVidia, “Nvidia geforce gtx 680,” tech. rep., NVidia, 2012.
- [121] NVidia, “Kepler tm gk110,” tech. rep., NVidia, 2012.
- [122] J. Jeffers and J. Reinders, *Intel Xeon Phi Coprocessor High Performance Programming*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st ed., 2013.
- [123] K. Gregory and A. Miller, *C++ AMP: Accelerated Massive Parallelism with Microsoft Visual C++*, Microsoft Press, 2012.
- [124] OpenACC, “The openacc application programming interface.”  
[http://www.openacc.org/sites/default/files/OpenACC.1.0\\_0.pdf](http://www.openacc.org/sites/default/files/OpenACC.1.0_0.pdf).
- [125] O. A. R. Board, “OpenMP application program interface.”  
<http://www.openmp.org/mp-documents/OpenMP3.1.pdf>.
- [126] B. Chapman, G. Jost, and R. van der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming*, The MIT Press, 2007.
- [127] H. Sutter, “The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software,” *Dr. Dobbs’s Journal* **30**(3), 2005.
- [128] R. Shams, P. Sadeghi, R. A. Kennedy, and R. I. Hartley, “A survey of medical image registration on multicore and the GPU,” *IEEE Signal Processing Mag.* **27**, pp. 50–60, Mar. 2010.
- [129] A. Eklund, P. Dufort, D. Forsberg, and S. M. LaConte, “Medical image processing on the GPU - past, present and future,” *Medical Image Analysis* **17**(8), pp. 1073–1094, 2013.
- [130] C. Sigg and M. Hadwiger, *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, ch. Fast Third-Order Texture Filtering, pp. 313–329. Addison-Wesley Professional, 2005.
- [131] D. Ruijters, B. M. ter Haar Romeny, and P. Suetens, “Efficient GPU-based texture interpolation using uniform B-splines,” *journal of graphics, gpu, and game tools* **13**(4), pp. 61–69, 2008.
- [132] T. J. Cullip and U. Neumann, “Accelerating volume reconstruction with 3D texture hardware,” tech. rep., University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 1994.
- [133] S.-Y. Guan and R. G. Lipes, “Innovative volume rendering using 3d texture mapping,” *Proc. SPIE* **2164**, pp. 382–392, 1994.
- [134] O. Wilson, A. VanGelder, and J. Wilhelms, “Direct volume rendering via 3d textures,” tech. rep., University of California at Santa Cruz, Santa Cruz, CA, USA, 1994.
- [135] B. Cabral, N. Cam, and J. Foran, “Accelerated volume rendering and tomographic reconstruction using texture mapping hardware,” in *VVS '94: Proceedings of the 1994 symposium on Volume visualization*, pp. 91–98, ACM Press, (New York, NY, USA), 1994.
- [136] J. Krüger and R. Westermann, “Acceleration techniques for GPU-based volume rendering,” in *Visualization, 2003. VIS 2003. IEEE*, pp. 287–292, oct. 2003.
- [137] F. Ino, Y. Kawasaki, T. Tashiro, Y. Nakajima, Y. Sato, S. Tamura, and K. Hagihara, “A parallel implementation of 2D-3D image registration for computer-assisted surgery,” *Int. J. Bioinformatics Res. Appl.* **2**(4), pp. 341–358, 2006.



- 
- [138] F. Ino, J. Gomita, Y. Kawasaki, and K. Hagihara, "A gpgpu approach for accelerating 2-d/3-d rigid registration of medical images," in *Parallel and Distributed Processing and Applications*, M. Guo, L. Yang, B. Di Martino, H. Zima, J. Dongarra, and F. Tang, eds., *Lecture Notes in Computer Science* **4330**, pp. 939–950, Springer, 2006.
- [139] W. Birkfellner, R. Seemann, M. Figl, J. Hummel, C. Ede, P. Homolka, X. Yang, P. Niederer, and H. Bergmann, "Wobbled splatting - a fast perspective volume rendering method for simulation of X-ray images from CT," *Phys Med Biol* **50**(9), pp. N73–N84, 2005.
- [140] D. Ruijters, B. M. ter Haar-Romeny, and P. Suetens, "GPU-accelerated digitally reconstructed radiographs," in *BioMED '08: Proceedings of the Sixth IASTED International Conference on Biomedical Engineering*, pp. 431–435, ACTA Press, (Anaheim, CA, USA), 2008.
- [141] R. Strzodka, M. Droske, and M. Rumpf, "Fast image registration in DX9 graphics hardware," *Journal of Medical Informatics and Technologies* **6**, pp. 43–49, Nov 2003.
- [142] A. Köhn, J. Drexl, F. Ritter, M. Koenig, and H.-O. Peitgen, "GPU accelerated image registration in two and three dimensions.," in *Bildverarbeitung für die Medizin, Informatik Aktuell*, pp. 261–265, Springer, 2006.
- [143] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence* **17**(1-3), pp. 185–203, 1981.
- [144] G. C. Sharp, N. Kandasamy, H. Singh, and M. Folkert, "GPU-based streaming architectures for fast cone-beam CT image reconstruction and demons deformable registration," *Phys. Med. Biol* **52**(19), pp. 5771–5783, 2004.
- [145] N. Courty and P. Hellier, "Accelerating 3D non-rigid registration using graphics hardware," *International Journal of Image and Graphics* **8**(1), pp. 1–18, 2008.
- [146] P. Muyan-Özçelik, J. D. Owens, J. Xia, and S. S. Samant, "Fast deformable registration on the GPU: A CUDA implementation of demons," in *The 2008 International Conference on Computational Science and Its Applications*, pp. 223–233, ICCSA 2008, IEEE Computer Society, 2008.
- [147] S. S. Samant, J. Xia, P. Muyan-Özçelik, and J. D. Owens, "High performance computing for deformable image registration: Towards a new paradigm in adaptive radiotherapy," *Medical Physics* **35**, pp. 3546–3553, Aug 2008.
- [148] X. Gu, H. Pan, Y. Liang, R. Castillo, D. Yang, D. Choi, E. Castillo, A. Majumdar, T. Guerrero, and S. B. Jiang, "Implementation and evaluation of various demons deformable image registration algorithms on a GPU," *Physics in Medicine and Biology* **55**(1), p. 207, 2010.
- [149] C. Rezk-Salama, M. Scheuring, G. Soza, and G. Greiner, "Fast volumetric deformation on general purpose hardware," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware, HWWS '01*, pp. 17–24, ACM, (New York, NY, USA), 2001.
- [150] G. Soza, P. Hastreiter, M. Bauer, C. Rezk-Salama, C. Nimsy, and G. Greiner, "Intraoperative registration on standard PC graphics hardware," in *Bildverarbeitung für die Medizin*, pp. 334–337, 2002.
- [151] G. Soza, M. Bauer, P. Hastreiter, C. Nimsy, and G. Greiner, "Non-rigid registration with use of hardware-based 3D Bézier functions," in *MICCAI '02: Proceedings of the 5th International Conference on Medical Image Computing and Computer-Assisted Intervention-Part II*, pp. 549–556, Springer-Verlag, (London, UK), 2002.
- [152] O. Fluck, S. Aharon, and A. Khamene, "Efficient framework for deformable 2D-3D registration," in *Proceedings of SPIE Medical Imaging, Proceedings of SPIE Medical Imaging* **6918**, pp. 69181E–1–69181E–10, SPIE, 2008.

- [153] H. Chen, J. Hesser, and R. Männer, “Fast free-form volume deformation using inverse-ray-deformation,” in *VIIP*, pp. 163–168, ACTA Press, 2001.
- [154] B. Li, A. Young, and B. Cowan, “GPU accelerated non-rigid registration for the evaluation of cardiac function,” in *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2008*, D. Metaxas, L. Axel, G. Fichtinger, and G. Székely, eds., *Lecture Notes in Computer Science* **5242**, pp. 880–887, Springer, 2008.
- [155] D. Ruijters, B. M. ter Haar-Romeny, and P. Suetens, “Accuracy of GPU-based B-spline evaluation,” in *CGIM '08: Proceedings of the Tenth IASTED International Conference on Computer Graphics and Imaging*, pp. 117–122, ACTA Press, (Anaheim, CA, USA), 2008.
- [156] D. Ruijters, B. M. ter Haar-Romeny, and P. Suetens, “Efficient GPU-accelerated elastic image registration,” in *BioMED '08: Proceedings of the Sixth IASTED International Conference on Biomedical Engineering*, pp. 419–424, ACTA Press, (Anaheim, CA, USA), 2008.
- [157] M. Modat, G. R. Ridgway, Z. A. Taylor, M. Lehmann, J. Barnes, D. J. Hawkes, N. C. Fox, and S. Ourselin, “Fast free-form deformation using graphics processing units,” *Comput. Methods Prog. Biomed.* **98**, pp. 278–284, Jun 2010.
- [158] T. Schiwietz, J. Georgii, and R. Westermann, “Interactive model-based image registration,” in *Proceedings of Vision, Modeling and Visualization 2007*, pp. 213–221, 2007.
- [159] K. Noe, K. Tanderup, J. Lindegaard, C. Grau, and T. Sørensen, “GPU accelerated viscous-fluid deformable registration for radiotherapy,” *Studies in Health Technology and Informatics* **132**, pp. 327–332, 2008.
- [160] T. ur Rehman, E. Haber, G. Pryor, J. Melonakos, and A. Tannenbaum, “3D nonrigid registration via optimal mass transport on the GPU,” *Medical Image Analysis* **13**(6), pp. 931–940, 2009.
- [161] T. ur Rehman, G. Pryor, J. Melonakos, and A. Tannenbaum, “Multi-resolution 3D nonrigid registration via optimal mass transport on the GPU,” in *Proceedings of Computational Biomechanics for Medicine-II*, pp. 122–132, Med Image Comput Comput Assist Interv. MICCAI 2007, Oct 2007.
- [162] A. Ruiz, M. Ujaldon, L. Cooper, and K. Huang, “Non-rigid registration for large sets of microscopic images on graphics processors,” *J. Signal Process. Syst.* **55**(1-3), pp. 229–250, 2009.
- [163] NVidia, “CUDA SDK.” <http://developer.nvidia.com/gpu-computing-sdk>.
- [164] R. Shams and N. Barnes, “Speeding up mutual information computation using NVIDIA CUDA hardware,” in *DICTA '07: Proceedings of the 9th Biennial Conference of the Australian Pattern Recognition Society on Digital Image Computing Techniques and Applications*, pp. 555–560, IEEE Computer Society, (Washington, DC, USA), 2007.
- [165] Y. Lin and G. Medioni, “Mutual information computation and maximization using GPU,” in *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*, pp. 1–6, June 2008.
- [166] W.-H. Cheng and C.-C. Lu, “Acceleration of medical image registration using graphics process units in computing normalized mutual information,” in *Proceedings of the 2009 Fifth International Conference on Image and Graphics, ICIG '09*, pp. 814–818, IEEE Computer Society, (Washington, DC, USA), 2009.
- [167] C. Guetter, C. Xu, F. Sauer, and J. Hornegger, “Learning based non-rigid multi-modal image registration using Kullback-Leibler divergence,” in *Proceedings of the 8th international conference on Medical image computing and computer-assisted intervention - Volume Part II, MICCAI'05*, pp. 255–262, Springer, 2005.

- 
- [168] M. Grabner, T. Pock, T. Gross, and B. Kainz, "Automatic differentiation for GPU-accelerated 2D/3D registration," in *Advances in Automatic Differentiation*, C. H. Bischof, H. M. Bücker, P. Hovland, U. Naumann, and J. Utke, eds., *Lecture Notes in Computational Science and Engineering* **64**, pp. 259–269, Springer, Aug 2008.
- [169] P. Chinnadurai, "Knowledge-driven image registration for accurate attenuation correction in hybrid spect/ct scanners - a clinical validation study," Master's thesis, Indian Institute of Technology, Kharagpur, India, May 2008.
- [170] C. Güttler, *Statistical Intensity Prior Models with Applications in Multimodal Image Registration*. PhD thesis, Universität Erlangen-Nürnberg, Erlangen, Germany, Jan 2011.
- [171] M. E. Leventon and W. E. L. Grimson, "Multi-modal volume registration using joint intensity distributions," in *Proceedings of the First International Conference on Medical Image Computing and Computer-Assisted Intervention, MICCAI '98*, pp. 1057–1066, Springer, 1998.
- [172] R. Gan, J. Wu, A. Chung, S. Yu, and I. Wells, WilliamM., "Multiresolution image registration based on Kullback-Leibler distance," in *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2004*, C. Barillot, D. Haynor, and P. Hellier, eds., *Lecture Notes in Computer Science* **3216**, pp. 599–606, Springer Berlin Heidelberg, 2004.
- [173] D. Cremers, C. Guetter, and C. Xu, "Nonparametric priors on the space of joint intensity distributions for non-rigid multi-modal image registration," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, **2**, pp. 1777–1783, 2006.
- [174] C. Guetter, M. Wacker, C. Xu, and J. Hornegger, "Registration of cardiac SPECT/CT data through weighted intensity co-occurrence priors," in *Proceedings of the 10th international conference on Medical image computing and computer-assisted intervention - Volume Part I, MICCAI'07*, pp. 725–733, Springer, 2007.
- [175] A. N. Tikhonov and V. Y. Arsenin, *Solutions of Ill-Posed Problems*, V. H. Winston & Sons, Washington, D.C.: John Wiley & Sons, New York, 1977.
- [176] G. Hermosillo, C. Chef d'Hotel, and O. Faugeras, "Variational methods for multimodal image matching," *International Journal of Computer Vision* **50**, pp. 329–343, 2002.
- [177] E. Parzen, "On estimation of a probability density function and mode," *The Annals of Mathematical Statistics* **33**(3), pp. 1065–1076, 1962.
- [178] R. Deriche, "Separable recursive filtering for efficient multi-scale edge detection," pp. 18–23, 1987.
- [179] C. Woolley, "GPU program optimization," in *GPU Gems 2 - Programming Techniques for High-Performance Graphics and General-Purpose Computation*, M. Pharr, ed., ch. 35, pp. 557–571, Addison-Wesley, 2005.
- [180] ARB, "EXT\_histogram." <http://opengl.org/registry/specs/EXT/histogram.txt>.
- [181] P. Hastreiter and T. Ertl, "Integrated registration and visualization of medical image data," in *Proceedings of Computer Graphics International (CGI)*, pp. 78–85, 1998.
- [182] M. Teßmann, C. Eisenacher, F. Enders, M. Stamminger, and P. Hastreiter, "GPU accelerated normalized mutual information and B-Spline transformation," in *Proceedings of the Eurographics Workshop on Visual Computing for Biomedicine (EG VCBM)*, pp. 117–124, The Eurographics Association, 2008.
- [183] O. Fluck, S. Aharon, D. Cremers, and M. Rousson, "Gpu histogram computation," in *ACM SIGGRAPH 2006 Research posters, SIGGRAPH '06*, ACM, (New York, NY, USA), 2006.
- [184] T. Scheuermann and J. Hensley, "Efficient histogram generation using scattering on GPUs," in *Proceedings of the 2007 symposium on Interactive 3D graphics and games, I3D '07*, pp. 33–37, ACM, (New York, NY, USA), 2007.

- [185] V. Podlozhnyuk, "Histogram calculation in CUDA," tech. rep., NVidia, 2007.
- [186] R. Shams and R. A. Kennedy, "Efficient histogram algorithms for NVIDIA CUDA compatible devices," in *Proc. Int. Conf. on Signal Processing and Communications Systems (ICSPCS)*, pp. 418–422, (Gold Coast, Australia), Dec 2007.
- [187] S. Chen, J. Qin, Y. Xie, W.-M. Pang, and P.-A. Heng, "Cuda-based acceleration and algorithm refinement for volume image registration," in *BioMedical Information Engineering, 2009. FBIE 2009. International Conference on Future*, pp. 544–547, 13-14 2009.
- [188] T. Brosch and R. Tam, "A self-optimizing histogram algorithm for graphics card accelerated image registration," in *Medical Image Computing and Computer Assisted Intervention (MICCAI) Grid Workshop*, pp. 35–44, 2009.
- [189] R. Shams, P. Sadeghi, R. A. Kennedy, and R. Hartley, "Parallel computation of mutual information on the GPU with application to real-time registration of 3D medical images," *Computer Methods and Programs in Biomedicine* **99**, pp. 133–146, Aug. 2010.
- [190] N. Satish, M. Harris, and M. Garland, "Designing efficient sorting algorithms for manycore GPUs," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pp. 1–10, 23-29 2009.
- [191] J. Radon, "Über die Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten," *Akad. Wiss.* **69**, pp. 262–277, 1917.
- [192] J. Radon, "On the determination of functions from their integral values along certain manifolds," *Medical Imaging, IEEE Transactions on* **5**, pp. 170–176, Dec 1986.
- [193] F. Natterer, *The Mathematics of Computerized Tomography*, SIAM,, 2001.
- [194] R. Gordon, R. Bender, and G. T. Herman, "Algebraic reconstruction techniques (art) for three-dimensional electron microscopy and X-ray photography," *Journal of Theoretical Biology* **29**(3), pp. 471–481, 1970.
- [195] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B* **39**(1), pp. 1–38, 1977.
- [196] L. A. Shepp and Y. Vardi, "Maximum likelihood reconstruction for emission tomography," *Medical Imaging, IEEE Transactions on* **1**, pp. 113–122, oct. 1982.
- [197] H. Hudson and R. Larkin, "Accelerated image reconstruction using ordered subsets of projection data," in *IEEE Transactions on Medical Imaging*, **13**, pp. 601–609, IEEE, Dec 1994.
- [198] G. Zeng and G. Gullberg, "Unmatched projector/backprojector pairs in an iterative reconstruction algorithm," *IEEE Transactions on Medical Imaging* **19**, pp. 548–555, May 2000.
- [199] P. Lacroute and M. Levoy, "Fast volume rendering using a shear-warp factorization of the viewing transformation," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques, SIGGRAPH '94*, pp. 451–458, ACM, (New York, NY, USA), 1994.
- [200] L. Westover, "Footprint evaluation for volume rendering," in *Proceedings of the 17th annual conference on Computer graphics and interactive techniques, SIGGRAPH '90*, pp. 367–376, ACM, (New York, NY, USA), 1990.
- [201] P. Sabella, "A rendering algorithm for visualizing 3D scalar fields," in *Proceedings of the 15th annual conference on Computer graphics and interactive techniques, SIGGRAPH '88*, pp. 51–58, ACM, (New York, NY, USA), 1988.
- [202] S. Metzler, J. Bowsher, M. Smith, and R. Jaszczak, "Analytic determination of pinhole collimator sensitivity with penetration," *IEEE Transactions on Medical Imaging* **20**, pp. 730–741, Aug 2001.

- 
- [203] K. Mueller and R. Yagel, "Rapid 3d cone-beam reconstruction with the simultaneous algebraic reconstruction technique (SART) using 2D texture mapping hardware," *IEEE Trans. Med. Imaging* **19**(12), pp. 1227–1237, 2000.
- [204] K. Chidlow and T. Möller, "Rapid emission tomography reconstruction," in *VG '03: Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume graphics*, pp. 15–26, ACM Press, (New York, NY, USA), 2003.
- [205] Z. Wang, G. Han, T. Li, and Z. Liang, "Speedup OS-EM image reconstruction by pc graphics card technologies for quantitative SPECT with varying focal-length fan-beam collimation," *IEEE Transactions on Nuclear Science* **52**, pp. 1274–1280, Oct 2005.
- [206] T. Schiwietz, S. Bose, J. Maltz, and R. Westermann, "A fast and high-quality cone beam reconstruction pipeline using the GPU," in *Proceedings of SPIE Medical Imaging 2006*, SPIE, (San Diego, CA), Feb 2007.
- [207] F. Xu and K. Mueller, "Real-time 3D computed tomographic reconstruction using commodity graphics hardware," *Physics in Medicine and Biology* **52**, pp. 3405–3419, Jul 2007.
- [208] C. Bai, G. L. Zeng, G. Gullberg, F. DiFilippo, and S. Miller, "Slab-by-slab blurring model for geometric point response correction and attenuation correction using iterative reconstruction algorithms," in *IEEE Transactions on Nuclear Science*, **45**, pp. 2168–2173, IEEE, Aug 1998.
- [209] A. K. Buck, S. Nekolla, S. Ziegler, A. Beer, B. J. Krause, K. Herrmann, K. Scheidhauer, H.-J. Wester, E. J. Rummeny, M. Schwaiger, and A. Drzezga, "SPECT/CT," *J Nucl Med* **49**, pp. 1305–1319, Aug 2008.
- [210] J. Czernin, M. R. Benz, and M. S. Allen-Auerbach, "PET/CT imaging: The incremental value of assessing the glucose metabolic phenotype and the structure of cancers in a single examination," *Eur J Radiol* **73**, pp. 470–480, Mar 2010.
- [211] G. Lyman, A. Giuliano, M. Somerfield, A. B. 3rd, D. Bodurka, H. Burstein, A. Cochran, H. C. 3rd, S. Edge, S. Galper, J. Hayman, T. Kim, C. Perkins, D. Podoloff, V. Sivasubramaniam, R. Turner, R. Wahl, D. Weaver, A. Wolff, and E. Winer, "American Society of Clinical Oncology guideline recommendations for sentinel lymph node biopsy in early-stage breast cancer," *J Clin Oncol* **23**, pp. 7703–20, 2005.
- [212] S. Vidal-Sicart and R. Valdés Olmos, "Sentinel node mapping for breast cancer: current situation.," *J Oncol* **2012**, p. 361341, 2012.
- [213] M. Baumhauer, M. Feuerstein, H.-P. Meinzer, and J. Rassweiler, "Navigation in endoscopic soft tissue surgery: perspectives and limitations," *J Endourol* **22**, pp. 751–766, Apr 2008.
- [214] A. Khamene, P. Bloch, W. Wein, M. Svatos, and F. Sauer, "Automatic registration of portal images and volumetric {CT} for patient positioning in radiation therapy," *Medical Image Analysis* **10**(1), pp. 96 – 112, 2006.
- [215] D. Gobbi, R. Comeau, B. Lee, and T. Peters, "Integration of intra-operative 3d ultrasound with pre-operative mri for neurosurgical guidance," in *Engineering in Medicine and Biology Society, 2000. Proceedings of the 22nd Annual International Conference of the IEEE*, **3**, pp. 1738–1740 vol.3, 2000.
- [216] A. Schnelzer, A. Ehlerding, C. Blümel, A. Okur, K. Scheidhauer, S. Paepke, and M. Kiechle, "Showcase of intraoperative 3D imaging of the sentinel lymph node in a breast cancer patient using the new freehand SPECT technology," *Breast Care* **7**(6), pp. 484–486, 2012.
- [217] C. Bluemel, A. Schnelzer, A. Okur, A. Ehlerding, S. Paepke, K. Scheidhauer, and M. Kiechle, "Freehand spect for image-guided sentinel lymph node biopsy in breast cancer," *European Journal of Nuclear Medicine and Molecular Imaging* **40**(11), pp. 1656–1661, 2013.
-

- [218] A. Rieger, J. Saeckl, B. Belloni, R. Hein, A. Okur, K. Scheidhauer, T. Wendler, J. Traub, H. Friess, and M. Martignoni, “First experiences with navigated Radio-Guided surgery using freehand SPECT,” *Case Reports in Oncology* **4**, pp. 420–425, 2011.
- [219] D. Heuveling, K. Karagozoglu, A. van Schie, S. van Weert, A. van Lingen, and R. de Bree, “Sentinel node biopsy using 3d lymphatic mapping by freehand spect in early stage oral cancer: a new technique,” *Clinical Otolaryngology* **37**(1), pp. 89–90, 2012.
- [220] T. Wendler, A. Hartl, T. Lasser, J. Traub, F. Daghighian, S. Ziegler, and N. Navab, “Towards intra-operative 3D nuclear imaging: Reconstruction of 3D radioactive distributions using tracked gamma probes,” in *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2007*, N. Ayache, S. Ourselin, and A. Maeder, eds., *Lecture Notes in Computer Science* **4792**, pp. 909–917, Springer, 2007.
- [221] T. Wendler, K. Herrmann, A. Schnelzer, T. Lasser, J. Traub, O. Kutter, A. Ehlerding, K. Scheidhauer, T. Schuster, M. Kiechle, M. Schwaiger, N. Navab, S. Ziegler, and A. Buck, “First demonstration of 3-d lymphatic mapping in breast cancer using freehand spect,” *European Journal of Nuclear Medicine and Molecular Imaging* **37**(8), pp. 1452–1461, 2010.
- [222] A. Hartl, D. I. Shakir, R. Kojchev, N. Navab, S. Ziegler, and T. Lasser, “Freehand SPECT reconstructions using look up tables,” in *Proceedings of SPIE Medical Imaging Conference (SPIE2012) 2012*, (San Diego, USA), February 2012.
- [223] N. Nikolaidis and I. Pitas, *3-D Image Processing Algorithms*, John Wiley and Sons, 2001.
- [224] S. Jan, G. Santin, D. Strul, S. Staelens, K. Assié, D. Autret, S. Avner, R. Barbier, M. Bardiés, P. M. Bloomfield, D. Brasse, V. Breton, P. Bruyndonckx, I. Buvat, A. F. Chatziioannou, Y. Choi, Y. H. Chung, C. Comtat, D. Donnarieix, L. Ferrer, S. J. Glick, C. J. Groiselle, D. Guez, P.-F. Honore, S. Kerhoas-Cavata, A. S. Kirov, V. Kohli, M. Koole, M. Krieguer, D. J. van der Laan, F. Lamare, G. LARGERON, C. Lartizien, D. Lazaro, M. C. Maas, L. Maigne, F. Mayet, F. Melot, C. Merheb, E. Pennacchio, J. Perez, U. Pietrzyk, F. R. Rannou, M. Rey, D. R. Schaart, C. R. Schmidtlein, L. Simon, T. Y. Song, J.-M. Vieira, D. Visvikis, R. V. de Walle, E. Wiërs, and C. Morel, “Gate: a simulation toolkit for PET and SPECT,” *Physics in Medicine and Biology* **49**(19), p. 4543, 2004.
- [225] J. Gardiazabal, T. Reichl, A. Okur, T. Lasser, and N. Navab, “First flexible robotic intra-operative nuclear imaging for image-guided surgery,” in *Information Processing in Computer-Assisted Interventions*, D. Barratt, S. Cotin, G. Fichtinger, P. Jannin, and N. Navab, eds., *Lecture Notes in Computer Science* **7915**, pp. 81–90, Springer Berlin Heidelberg, 2013.
- [226] J. Vogel, T. Lasser, J. Gardiazabal, and N. Navab, “Trajectory optimization for intra-operative nuclear tomographic imaging,” *Medical Image Analysis* **17**(7), pp. 723–731, 2013.
- [227] P. Matthies, J. Gardiazabal, A. Okur, J. Vogel, T. Lasser, and N. Navab, “First use of mini gamma cameras for intra-operative robotic SPECT reconstruction,” in *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2013*, K. Mori, I. Sakuma, Y. Sato, C. Barillot, and N. Navab, eds., *Lecture Notes in Computer Science* **8149**, pp. 163–170, Springer Berlin Heidelberg, 2013.